Innovation Report

Innovative Solutions for Automotive Embedded Software Development

Submitted in partial fulfilment of the requirements for the degree of Doctor of Engineering (EngD)

By Alexandros Mouzakitis, 1136297

30 November 2016

Academic Supervisors: Professor P. A. Jennings, Dr. J. Marco, G. Dhadyalla

Industrial Supervisor: G. Lancaster

To my beautiful children Alexia and Adam

Abstract

Embedded software is shaping and influencing our world and it is unimaginable to realise day to day life without it. Since the introduction of the first Electronic Control Unit (ECU) in the 1970s, the automotive industry has seen a substantial increase of embedded software in vehicles. The use of embedded software in the automotive industry has led to a significant increase in the number and complexity of different vehicle systems, features and functions. This level of complexity drives premium vehicles with no fewer than 70 ECUs interconnected by more than five on-board network systems such as Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST), FlexRay and Ethernet.

In a typical automotive development process, the main challenge for the engineers is to uncover as many failure modes and/or software defects as possible during the early stages of the vehicle programme. During the early phases of the development, failure modes and/or software defects are difficult to uncover but easy and inexpensive to fix. During the latter phases of the development, failure modes and/or software defects are easy to uncover since the final product has been built. At this stage, failure modes and/or software defects are hard and expensive to fix as changes required in the embedded software.

The aim of this research was to develop and deploy innovative solutions in order to shift failure modes and/or software defects detection early in automotive product development. The initial research work was conducted through an analysis of failure modes and/or software defects found during a typical Jaguar Land Rover (JLR) vehicle programme development. This preliminary work also then focused on supplier base capability for automotive embedded software development. The research findings from the internal and external analysis, together with the literature review on best practice have driven the development of four solutions.

A process called Model-based Product Engineering (MBPE) was created and deployed within JLR. The MBPE process brings together model-based development and other development processes in a standardised form. A new generic Design Verification Interface (DVI) for test exchange and traceability across all MBPE process levels was developed. The generic DVI eliminates or reduces redundant efforts of re-writing test cases and test scripts for automated testing. A semi-formal Standardised Design Verification Method (SDVM) was developed for defining test cases for all vehicle systems in a common template. The SDVM presents test cases as machine readable data and allows auto-generation of test scripts suitable for automated testing. An end-to-end solution called Platform Independent Test System was developed in order to integrate the MBPE, DVI and SDVM solutions. The proposed PITS supports all levels of system abstraction from the test case definition phase to the execution of automated scripts in both offline and real-time test environments.

Evaluation results have demonstrated a significant shift in the detection of failure modes and/or software defects towards the early phases of the product development. An early detection of more than 50% of failure modes and/or software defects was achieved. This is a substantial change from the previous state where embedded software validation was conducted only after supplier software release. Furthermore, results have shown a 40% reduction in engineering effort for test scripts creation and a five to tenfold reduction in engineering time for automated testing.

Declaration

I have read and understood the rules on cheating, plagiarism and appropriate referencing as outlined in my handbook and I declare that the work contained in this submission is my own, unless otherwise acknowledged.

…….

Alexandros Mouzakitis

30 November 2016

Acknowledgements

I would like to thank Jaguar Land Rover (JLR) and the Engineering and Physical Sciences Research Council (EPSRC) for funding this project. I would also like to offer special thanks to my supervisors Professor Paul A. Jennings; Dr. James Marco, Gunwant Dhadyalla and Gerard Lancaster, for their time, support and guidance throughout this project.

Table of Contents

List of Tables

List of Figures

List of Acronyms and Abbreviations

AADL	Architecture and Analysis Description Language
ACC	Adaptive Cruise Control
ADL	Architecture Description Language
ASAM	Association for Standardisation of Automation and Measuring Systems
ATX	Automotive Test eXchange
AUTOSAR	Automotive Open System Architecture
BCM	Body Controller Module
BDD	Block Definition Diagram
CAN	Controller Area Network
CMMI	Capability Maturity Model Integration
CO_2	Carbon Dioxide
COTS	Commercial-Off-The-Shelf
DVI	Design Verification Interface
DVM	Design Verification Method
EAST-ADL	The Electronics Architecture and Software Technology - Architecture Description Language
ECU	Electronic Control Unit
EMS	Engine Management System
EPSRC	Engineering and Physical Sciences Research Council
FMI	Functional Mock-up Interface
FRs	Functional Requirements
GUI	Graphical User Interface
HIL	Hardware-in-the-Loop
HMI	Human Machine Interface
IEEE	Institute of Electrical and Electronics Engineers

ISO	International Organisation for Standardisation
JLR	Jaguar Land Rover
LIN	Local Interconnect Network
MARTE	Modelling and Analysis of Real-Time and Embedded systems
MBD	Model-based Development
MBPE	Model-based Product Engineering
MIL	Model-in-the-Loop
MOST	Media Oriented Systems Transport
NFRs	Non-Functional Requirements
ODX	Open Diagnostic data eXchange
OEM	Original Equipment Manufacturer
OMG	Object Management Group
OTX	Open Test sequence eXchange
PCS	Product Creation System
PEP	Product Evolution Process
PIT	Platform Independent Test
PITS	Platform Independent Test System
PoC	Proof of Concept
R&D	Research and Development
RCP	Rapid Control Prototyping
RE	Requirements Engineering
RTE	Run Time Environment
RTEA	Real-Time and Embedded Analysis
RTEM	Real-Time and Embedded Modelling
SDVM	Standardised Design Verification Method
SIG	Special Interest Group

SIL	Software-in-the-Loop
SoW	Statement of Work
SPICE	Software Process Improvement and Capability dEtermination
SysML	Systems Modelling Language
TCRM	Time and Concurrent Resource Modelling
UML	Unified Modelling Language
UML2	Unified Modelling Language 2
V&V	Verification and Validation
VoC	Voice of Customer
XML	eXtensible Markup Language
XSD	XML Schema Definition

1. Introduction

1.1. Background

Embedded software is influencing our world and it is difficult to envisage day to day activities without it (Ebert and Jones, 2009a). Over the last 20 years software has had an impact on embedded systems functionality. Furthermore the innovation of products has increased rapidly (Liggesmeyer and Trapp, 2009a). Embedded software can be found in many applications, such as mobile phones, medical equipment, energy generation, home appliances, aircrafts, ships, chemical plants and automotive electronics such as infotainment, engine, brakes and comfort systems (Lee, 2000), (Graaf *et al.*, 2003), (Ebert and Jones, 2009b), (Liggesmeyer and Trapp, 2009b), (Li and Malik, 2012), (Zhuang *et al.*, 2014). According to (Qian *et al.*, 2009) a typical embedded system consists of hardware and software parts. The hardware part includes a microprocessor or microcontroller and other components such as sensors and actuators. The embedded software part is the driving force of an embedded system and usually has processing and memory constraints. In the automotive industry most of the embedded software is executed in real-time. Real-time embedded software means that response time is critical since all scheduled tasks must be executed in a specified time period (Gai and Violante, 2016).

Introducing and managing in-vehicle embedded software is not new to the automotive industry (Lutz, 1993), (Bennett and Wennberg, 2005), (Sangiovanni-Vincentelli, 2000), (Navet and Simonot-Lion, 2008). Since the deployment of the first Electronic Control Unit (ECU) in the 1970s, the automotive industry has seen a trend in the increase of embedded software in vehicles, (Grimm, 2003), (Broy, 2006), (Henzinger and Sifakis, 2006), (Pretschner *et al.*, 2007), (Hanselmann, 2008), (Liggesmeyer and Trapp, 2009a), (Fürst, 2010), (Schulze *et al.*, 2012), (Maurer and Winner, 2013a), (Petrenko *et*

al., 2015a). The trend in the increase of embedded software has been more noticeable in the past 15 years due to ECU processing power and memory availability to the vehicle manufacturers and suppliers.

The complexity of automotive electronics with embedded software continuous to increase exponentially (Hanselmann, 2008), (Mossinger, 2010). According to (Polzer *et al.*, 2012a), (Buckl *et al.*, 2012) the amount of embedded software in vehicles within 30 years has increased from 0 to 10 million lines of code. At the beginning of this research this number was much higher, as a premium vehicle can reach up to 100 million lines of code (JLR-APC, 2012). Additionally, premium vehicles have no fewer than 70 ECUs interconnected by more than five network systems such as Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST), FlexRay and Ethernet (JLR-APC, 2012).

It is widely accepted in the automotive industry that embedded software in vehicles realise more than 2,000 functions and features can contribute 50%-70% of the development costs of an ECU (Grimm, 2003), (Yeaton, 2004), (Conrad *et al.*, 2005), (Broy, 2006), (Karsai, 2006), (Charette, 2009), (Mondragon *et al.*, 2009), (Gmehlich and Jones, 2013). Furthermore, up to 40% of the production cost of a vehicle is due to embedded software and electronics. Even in 2010, (Fürst, 2010) suggests, that 90% of all automotive innovations are driven by electronics and in-vehicle embedded software.

1.2. Vehicle features, functions, components and systems

A high level decomposition of a typical premium vehicle system architecture is shown in Figure 1. The vehicle embedded software architecture consists of a number of systems (very often referred as domains) such as body, infotainment, power supply,

safety, comfort, powertrain and hybrids. Each system consists of a number of ECUs (or components) such as Body Controller Module (BCM), Engine Management System (EMS) and Adaptive Cruise Control (ACC). Each component is made up of a number of functions designed to support vehicle features distributed across many components and systems. It is essential to state that the breakdown structure of the vehicle system architecture shown in Figure 1 is typical of all premium vehicles within the automotive industry (Navet and Simonot-Lion, 2008).

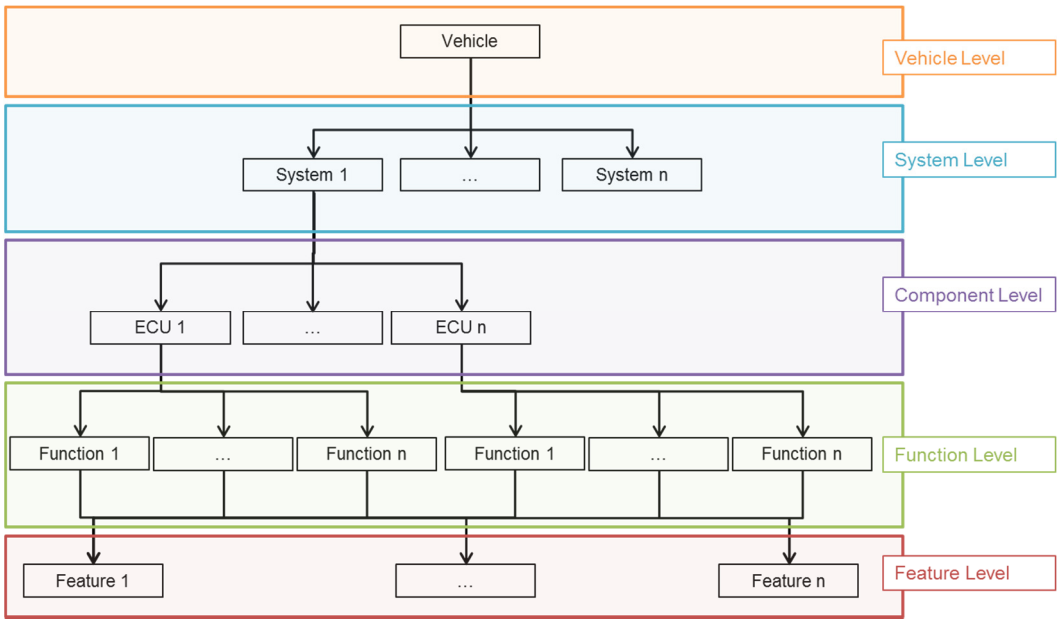


Figure 1. High level decomposition of a typical vehicle system architecture.

It is very common in the automotive industry to link vehicle features to customer needs and wants. According to (Navet and Simonot-Lion, 2008) a feature is a characteristic or trait in the broadest sense that an individual product instance of a product line may or may not have. In addition to that, a feature may not necessarily correspond to a system or component and it can be selected or deselected in the product configuration. Typical examples of a vehicle feature are auto-wiping, navigation, locking and auto-parking. The deployment of these features require embedded software development and

processes, methods and tools capable for robust and reliable product creation and a fast time to market.

The use of embedded software and new electronics within the automotive industry has led to a significant increase in the number and complexity of different vehicle features and functions. As mentioned earlier vehicle features are driven by customer expectations. The increase in the numbers of vehicle features realised by embedded software will continue to grow as customers demand higher quality, reliability and comfort. In addition to that, external factors such as legislation for CO_2 reduction and road safety will contribute significantly to the increase in automotive embedded software development and test. Automotive companies including Jaguar Land Rover (JLR) must create robust embedded software while managing the pressures of increased system complexity and reduced time to market. Furthermore, the control of the increased complexity of vehicle systems represents a core challenge as more and more automotive companies are undertaking embedded software development activities in house (Schulze *et al.*, 2012).

1.3. Vision of the project

In a typical automotive development process the main challenge of the engineers is to uncover as many failure modes and/or software defects as possible during the early stages of the vehicle programme. The IEEE Standard Glossary of Software Engineering Terminology (September, 1990) is used in this research project for the definitions of failure, failure mode, fault, error and defect. Details for these definitions are given in Submission 1.

An illustration of a typical product development timeline and the effort for finding and fixing failure modes and/or software defects is shown in Figure 2. During the early phases of the development, failure modes and/or software defects are difficult to uncover but easy and inexpensive to fix. At this early stage failure modes and/or software defects are difficult to uncover because only requirements and early concepts exist and the product has not been built yet. During the last phases of the development, failure modes and/or software defects are easy to uncover because the final product has been built. At this stage, failure modes and/or software defects are hard and expensive to fix as changes required in the software coding, hardware, tooling, verification and validation of the final product.

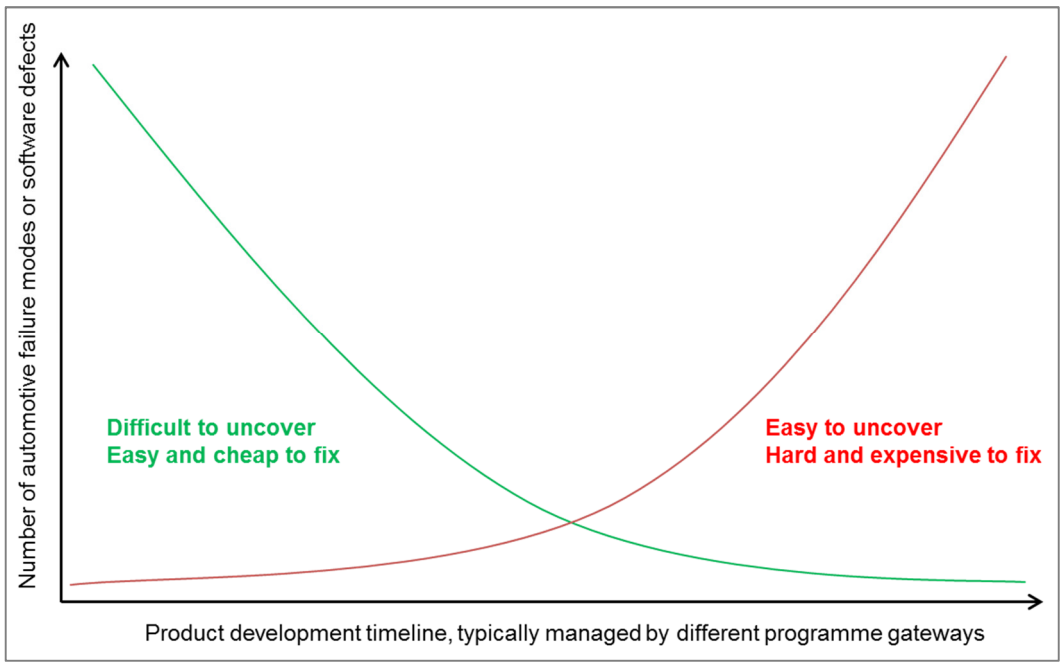


Figure 2. Failure modes and/or software defects detection points during a typical product development timeline.

Figure 3 shows the associated cost of repair of fixing a failure mode or a software defect introduced at a different stage of the product development including the stage at which a product is built and sold to customers (Lazic and Mastorakis, 2008). More

specifically in Figure 3 is shown that a failure mode or a software defect introduced in the requirements phase will cost approximately 368 times more to fix in the customer phase than in the phase in which was introduced. Correspondingly, it will cost 5 times more to fix in the design phase, 10 times more in the code phase, 50 times more in the test phase and 130 times more in the integration phase. A summary of the main consequences of late failure modes and/or software defects detection is given in Section 1.4.3.

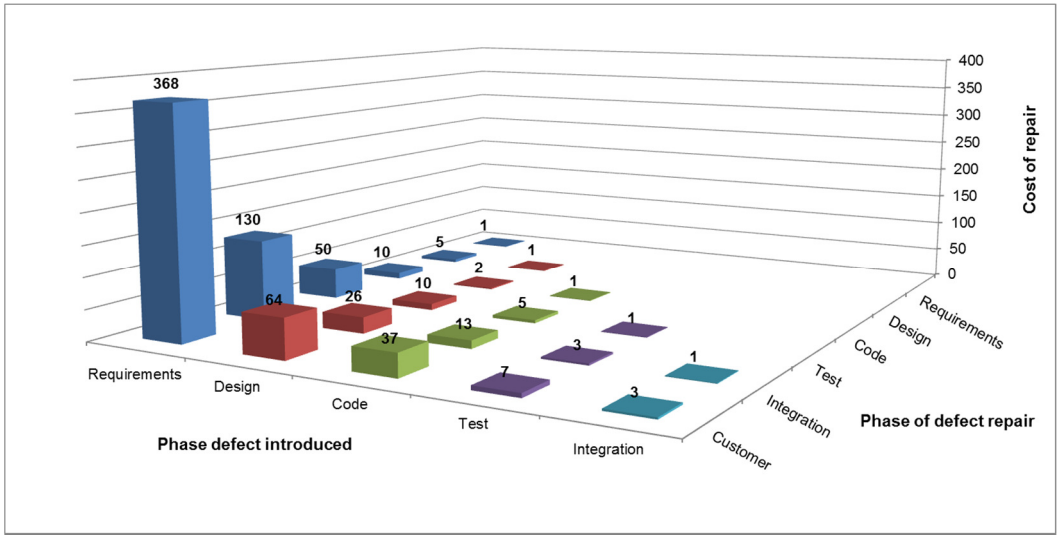


Figure 3. Failure modes and/or software defects introduction and repair phases during a typical product development timeline

The challenge of detecting failure modes and/or software defects during the early stages of the product development has led the automotive industry to research new and innovative ideas in the area of embedded software development and test (Murphy *et al.*, 2008), (Alemanni *et al.*, 2011), (Krogstie, 2012), (Shahbakhti *et al.*, 2012), (Chakraborty *et al.*, 2012a), (Goto, 2013), (Shahrokni and Feldt, 2013). As a result researchers come up with a number of specific research challenges that they believe, if solved, can potentially help to improve automotive embedded software development and test processes. As an example, (Hanselmann, 2008) identifies several research

challenges for embedded software in vehicles. These challenges include the need to improve hardware and software architectures for vehicles; reduce system complexity; improve software development processes; develop and deploy improved model-based software development processes; and integrate tool support for software development and automated testing.

However, the introduction of embedded software in vehicles is affecting all phases of automotive product development due to variability of ECUs and systems aiming to deliver customer intent functionality. A change in one function, ECU or a vehicle system can affect the whole product development and test (need for new hardware or re-tooling). These changes are driven from the need to support different vehicle markets as well as customer selectable features.

The vision of this project is to address the automotive embedded software development challenge of detecting failure modes and/or software defects at the early stage of the development process.

1.4. Motivation, research aim and objectives

The scope of this section is to present the motivation and the problem quantification for this research. The understanding and quantification of the problem was conducted through a real world automotive case study derived from Jaguar Land Rover (JLR). In addition to that the aim and objectives of this research project are presented.

1.4.1. Company background

The direct beneficiary of this research project is JLR and its supplier base. JLR is the UK's largest automotive manufacturing company, built around two iconic British vehicle brands. Land Rover, a manufacturer of premium all-wheel drive vehicles and Jaguar, manufacturer of premier luxury sports saloon and sports car marques. Under the ownership of Tata Motors Limited, JLR is employing approximately 26,000 people globally. In addition to that JLR supports more than ~190,000 UK jobs through the supply chain, dealer network and wider economy. Furthermore, JLR is the biggest UK investor in Research and Development (R&D) (~£3.5 Billion annual investment) in the manufacturing sector, ahead of British Aerospace and Rolls-Royce, and in the global top 100 for R&D spend (Jaguar Land Rover, 2014). More information related to the history of the company is given in Submission 1.

1.4.2. Motivation

As mentioned earlier the automotive industry has changed drastically in the last 20 years mainly due to the introduction and use of embedded software in vehicles. This change has also affected Jaguar Land Rover. JLR and the rest of the automotive industry are aiming to reduce product development cycles while having to introduce new and innovative customer features mainly driven by embedded software. Product development cycles are difficult to shorten if failure modes and/or software defects are

found late in product development. The motivation for this research was amplified and strengthened from the fact that many challenges associated to automotive embedded software development processes are yet to be resolved (Bosch and Eklund, 2012), (Chakraborty *et al.*, 2012b), (Lind and Heldal, 2012), (Mader *et al.*, 2012), (Polzer *et al.*, 2012b), (Broy *et al.*, 2013), (Studnia *et al.*, 2013), (Braun *et al.*, 2014a). The spectrum of these challenges is ranging from new processes, methods and tools for embedded software to new skills and organisational changes. Being able to fast develop new software features and integrate them with existing vehicle systems provides the automotive OEMs with a key competitive advantage and brand differentiation. New software features must be developed to meet the intent customer and system specification as well as being robust to noise factors such as system configuration and system updates.

The motivation for this research is driven from the increased complexity of automotive embedded software in vehicles and the drive to reduce product development cycles through early detection of failure modes and/or software defects.

In the following section a real world case study is presented in order to better understand and quantify the exact problem to be addressed by this research project.

1.4.3. Understanding and quantification of the problem

JLR use a specific product development system called Product Creation System (PCS) to design, develop and launch new vehicles into the market. The origin and details of JLR's PCS is given in Submission 1. JLR's PCS consists of a series of workstreams that allow development and progression from requirements, through design and

engineering, to validation, and ultimately to launch and mass production of vehicles. These workstreams are designed to ensure all aspects of the vehicle are fully compatible throughout the process with key synchronisation checkpoints at which progress is measured. JLR's PCS is mainly made of gateways, prototype builds and assembly plant builds. These are described in more detail in Submission 1. The PCS process has specific targets for failure mode and/or software defects detection. These targets are associated with specific gateways. A symbolic representation of a target is defined as *"X% of failure modes and/or software defects detected by Y gateway"*. If a target is missed then the first production intent prototype vehicle at the later gateways is at risk. The risk is associated with the late detection of failure modes and/or software defects causing late Electronic Control Unit (ECU) software and hardware changes. The specific JLR failure modes and/or software defects detection targets and the associated PCS gateways are given in Submission 1.

The purpose of this case study was to collect and analyse a recent JLR vehicle programme in order to greater understand and quantify the problem of late detection of failure modes and/or software defects during product development.

The key steps of the case study and analysis were as follows:

- Collect data (failure modes and/or software defects) from the start of the vehicle programme development phase through to mass production.
- Exclude from the analysis data related to mechanical failure modes and defects and focus the analysis on software related data only.
- Map collected data against PCS timing and investigate detection points for failure modes and/or software defects.

- Investigate severity ratings of failure modes and/or software defects.
- Conclude with problem identification and relation to this research project.

Data collected from a recent JLR vehicle programme are shown in Figure 4. The chosen vehicle programme was appropriate and representative of both Jaguar and Land Rover brands in terms of in-vehicle embedded software complexity, number of ECUs and number of on-board vehicle networks. Figure 4 depicts the number of failure modes and/or software defects found during a vehicle programme development cycle. A typical vehicle programme development cycle is ranging from 36 to 48 months depending on the system complexity. System complexity mainly is driven from the introduction of new vehicle features and technologies. Failure modes and/or software defects found are mapped against PCS gateways. In this case study the names of the gateways are anonymised in order to protect JLR's confidentiality. In addition to that the actual numbers of failure modes and/or software defects are not shown, again, due to JLR's confidentially.

In this case study the pattern and the trend of collected data are more important than the detail. The pattern is important because it can be linked to the associated programme gateways. Programme gateways hold key information such as anticipated introduction of hardware and software changes respectively. The trend of the data is equally important as several conclusions can be drawn from an increasing or decreasing trend towards mass production. The full details of this case study, including PCS gateways and raw data are provided in Submission 1.

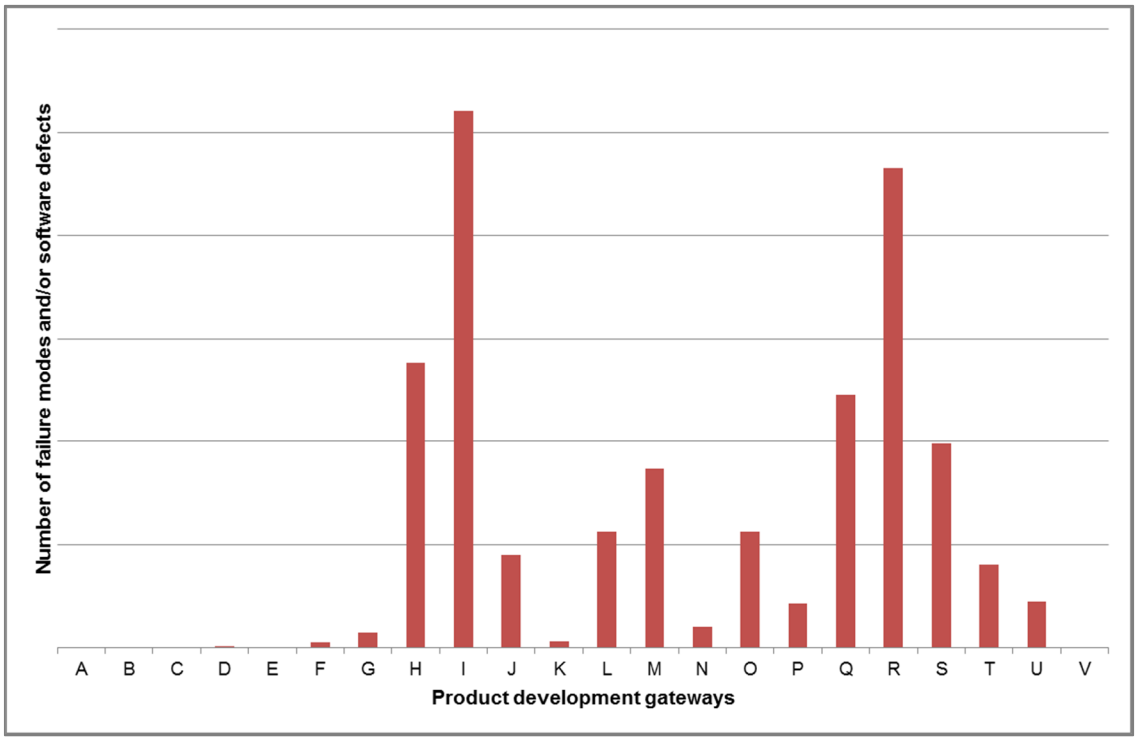


Figure 4. Failure modes and/or software defects detection points during a typical JLR vehicle programme development *(Due to confidentiality the actual numbers of failure modes and/or software defects, detection targets and gateways names are not shown).*

The data gathered in the study, as shown in Figure 4, suggest that the majority of failure modes and/or software defects detection occur near gateways I and R. Nearby gateways H and M, are programme development points where ECU hardware and vehicle prototypes are available to the engineers. At these phases (gateways I and R) of the product development, failure modes and/or software defects are easy to uncover but hard and expensive to fix, as was illustrated earlier in Figure 2. The symbolic representation of JLR's internal target defined earlier of *"X% of failure modes and/or software defects detected by Y gateway"* in this case study was not met (more details of this are given in Submission 1). Undoubtedly, failure modes and/or software defects are detected late in the product development process causing expensive engineering changes and reduced capability to design and develop vehicle features fast to market. The priority here for JLR product development was the timing of late detection rather

than the actual number of failure modes and/or software defects. This case study included data collected from both internal (JLR access only) and external (supplier access to JLR data) failure modes and/or software defects tracking systems. Data from both sources produced the same pattern and trend. More details on these tracking systems and gathered data are provided in Submission 1.

Failure modes and/or software defects typically are linked to either user/customer requirements or service requirements. As an example a loss of radio functionality is linked to user/customer requirement, whereas an issue with the embedded software download procedure is linked to service and vehicle maintenance requirement. Each failure mode and/or software defect is normally assigned to an issue severity number. The severity numbers used for this case study are shown in Table 1. Issues with severity number 1 have no customer impact. This is the case where embedded software meets the customer intent functionality but a further improvement can be made. Severity 2, 3 and 4 relate to issues that have customer impact and their associated detection point in the product development process (typically programme gateway). These issues will have to be addressed in order to ensure that there is not customer impact and therefore mass production can go ahead with robust vehicle build.

A histogram of the severity rating of the failure modes and/or software defects profile shown in Figure 4 is given in Figure 5. Most failure modes and/or software defects have been assigned with severity rating of 3. The findings of this research case study illustrate the antecedents and consequences of late detection of failure modes and/or software defects on the quality of the embedded software. The detection of these failure modes and/or software defects must be shifted to the earlier phases of the product development process. The priority is not necessary to remove severity ratings

4 or 3 but to enable the detection of all of them at the early stage of the vehicle programme.

The findings of this case study confirms the need for new solutions within the automotive industry in order to address real world problems linked to embedded software development and test.

Severity	Definition
1	Issue with no customer impact
2	Moderate issue plus gateway detected.
3	Significant issue plus gateway detected.
4	Major issue plus gateway detected.
5	Not used

Table 1. Failure modes and/or software defects severity definition.

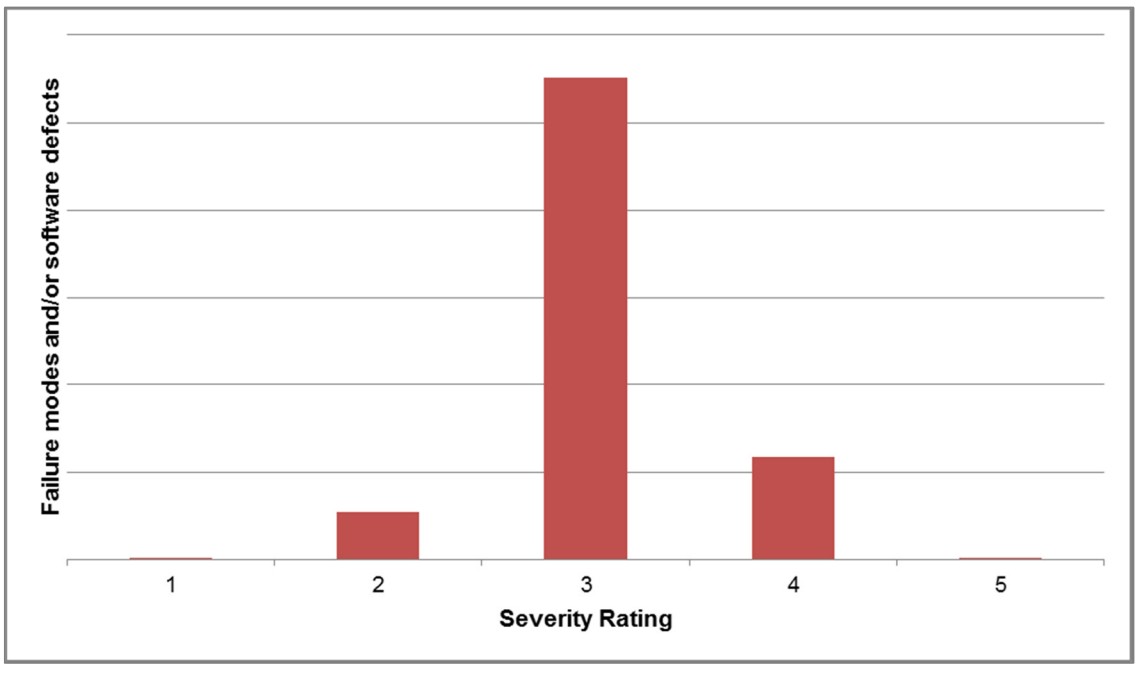


Figure 5. Severity ratings for the failure modes and/or software defects *(Due to confidentiality the actual numbers of failure modes and/or software defects are not shown)*.

It is important to note that all failure modes and/or software defects depicted in Figure 4 have been fixed prior to the first production intent vehicle build.

In conclusion, the **problem statement** resulting from this case study and analysis can be summarised as follows:

Relatively to an internal target too many failure modes and/or software defects associated with embedded software are found late in the automotive product development process.

The main consequences of late failure modes and/or software defects detection can be summarised as follows:

- Expensive ECU hardware and software changes.
- Unnecessary engineering resource utilisation resulting in delays to start new vehicle programmes.
- Risk of not meeting vehicle mass production date.
- Rick of failure modes and/or software defects affecting vehicle quality and hence leading to customer dissatisfaction.
- Risk for unnecessary warranty costs.

1.4.4. Research aim and objectives

The narrowed topic of research includes only ECU embedded software and excludes other type of software such as off-board, IT, product operations, logistics, manufacturing or service support related software. In addition to that the narrowed

topic excludes failure modes and/or software defects driven by mechanical designs and other related engineering activities such as aerodynamics and body structures.

The **research aim** of this research project was:

To shift failure modes and/or software defects detection early[1] in the automotive product development process.

The definition of the aim of this research project led to the following **research objectives**.

1. Assess how the integration of JLR and supplier base can influence the quality of automotive embedded software. Capture and analyse automotive supplier base capability for embedded software development and test.
2. Capture best practice for automotive embedded software development and identify areas for potential improvement.
3. Develop, integrate and implement solutions for embedded software in order to shift failure modes and/or software defects detection early in the automotive product development process.
4. Evaluate proposed solutions for automotive embedded software development using real world case studies.
5. Summarise research findings, identify business benefits and areas for future research.

[1] The definition of early detection is related to an internal JLR target.

1.5. Portfolio structure

The structure of the portfolio is shown in Figure 6. The portfolio consists of 6 individual submissions numbered 1 through 6. The portfolio also includes this innovation report and the personal profile. The recommended reading order is depicted in Figure 6. The statement of innovation and personal development are advised to be read first followed by the flow of individual submissions.

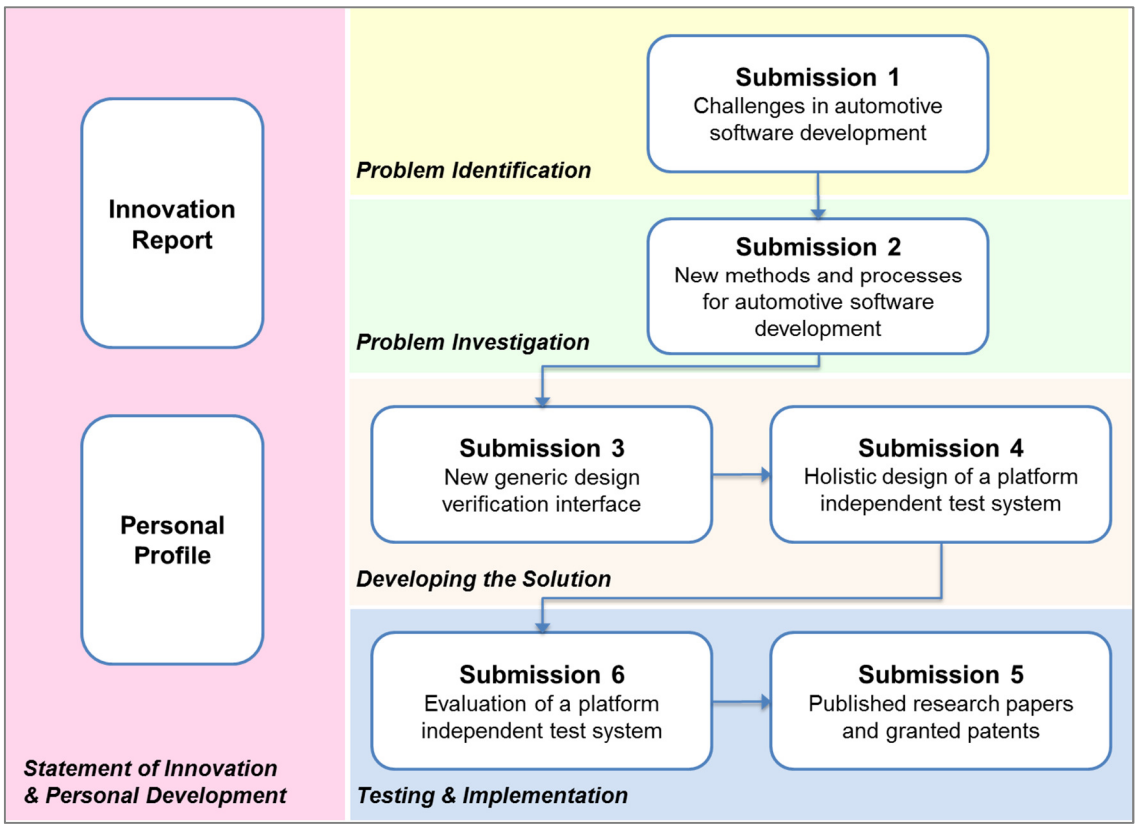


Figure 6. Portfolio structure and reading order of submissions.

The submissions cover all stages of the research and development starting from the problem identification and investigation leading to developing the solution and implementation. Each submission is now briefly reviewed in terms of its aim and how it links to the overall research story.

- **Submission 1 – Challenges in automotive software development**

  The aim of this submission was to present the main challenges linked to the automotive embedded software development. These challenges led to the definition of the research question. A case study from a typical JLR vehicle programme was undertaken in order to analyse where failures modes and/or software defects are found during the product development. The analysis moved beyond JLR into the supplier base in order to underpin the embedded software development challenges within the wider automotive industry. This submission concluded with the research aim and objectives of this project.

- **Submission 2 – New methods and processes for automotive software development**

  This submission focused on the creation of integrated methods and process for automotive embedded software development. The state-of-the-art was researched and as a result a new innovative process called Model-based Product Engineering (MBPE) was proposed. Two new engineering standards and two new design rules were introduced in order to enable effective MBPE deployment within JLR and its supplier base. Evaluation results were analysed and discussed against a set of evaluation criteria specified for the MBPE process. The evaluation of the MBPE process focused on its detection capability for failure modes and/or software defects at the early stages of the product development as well as on the impact for reduced verification and validation lead-times. Key benefits resulted from the deployment of the MBPE process within JLR were presented. The research outcome from this submission led directly onto Submission 3.

- **Submission 3 – New generic design verification interface**

  Submission 2 confirmed the need for research and development in the area of test exchange and traceability across all Model-based Product Engineering (MBPE) process levels. As a result, Submission 3 focused on the research and development of a new generic Design Verification Interface (DVI). Unified Modelling Language (UML) and eXtensible Markup Language (XML) were used to define the specification of the generic DVI. A set of evaluation criteria was drawn to assess DVI capability against state-of-the-art and best practice within the automotive industry. The research outcome from this submission led onto Submission 4.

- **Submission 4 – Holistic design of a platform independent test system**

  The research outcomes from Submissions 1, 2 and 3 led to the need to develop an end-to-end solution from the definition phase of test cases to the auto-generation of test scripts suitable for the execution of automated testing in both offline and real-time environments. As a result, Submission 4 focused on the holistic design and creation of such a solution called Platform Independent Test System (PITS). This end-to-end innovative solution pledged the potential to shift failure modes and/or software defects to the early stages of the product development. In this submission a new and innovative semi-formal Standardised Design Verification Method (SDVM) template was proposed. The aim of the new template was to allow test cases from all vehicle systems to be written in a standardised form and presented as machine readable data. Best practice in the area of requirements specification and verification was evaluated through a literature review on the use of informal, semi-formal and formal approaches. The submission presented how the SDVM template and the Design Verification Interface (DVI) were integrated in order to drive a holistic test solution across all Model-based Product Engineering (MBPE)

process levels. This submission presented a comprehensive review and assessment of existing DVM methods and test capability within JLR. The PIT system was briefly introduced and the main steps for test scripts auto-generation and execution in different test targets were highlighted. The holistic design of PITS facilitated the evaluation of the solutions against the project's main aim and objectives. Results are presented in Submission 6.

- **Submission 5 – Published research papers and granted patents**

  The aim of Submission 5 was to cover published research papers and granted patents, explaining their purpose and value to the company. The research outcomes thus far produced two research papers and two granted patents. Paper publications focused on ontology of the proposed Model-based Product Engineering (MBPE) process and its deployment within JLR. Both patents covered new methods for vehicle test and validation applicable to some of the levels of the proposed MBPE process.

- **Submission 6 – Evaluation of a platform independent test system**

  This submission covered the evaluation of the proposed Platform Independent Test System (PITS), explaining its benefits and value to the company. Evaluation results were analysed and conclusions were drawn in order to understand the impact of the research. The main focus of the evaluation was on engineering effort and time required to define test cases and execute automated test scripts in both offline and real-time test environment and test targets. Reduction in engineering effort and time has the potential to shift detection of failure modes and/or software defects to the early stages of the product development. The evaluation of PITS included a

comparison against commonly used Commercial off the Shelf (COTS) test automation tools within the automotive industry.

1.6. Outline of the innovation report

This innovation report is outlined as follows:

Section 2 presents the need for improvement in the area of automotive embedded software development. The research on best practice and literature review includes a supplier base capability assessment in order to identify potential root causes in late detection of failure modes and/or software defects during product development. The literature review on automotive embedded software development includes research on existing development processes, architecture description languages, software and quality standards, model and test exchange standards as well as approaches for test specification and test scripts auto-generation.

The research methodology for this project is explained in Section 3.

The development of solutions for automotive embedded software is presented in Section 4. Section 4 describes four key innovative solutions which directly address the research aim and objectives of this project. These are: Model-based Product Engineering (MBPE) process; generic Design Verification Interface (DVI); Standardised Design Verification Method (SDVM); Platform independent Test System (PITS). All four solutions are integrated in order to provide a holistic approach to the problem of late detection of failure modes and/or software defects during product development.

Section 5 presents an evaluation of the research. Evaluation criteria for the proposed solutions are derived and evaluation results are analysed in order to assess the impact and benefits of this research. The majority of the evaluation is driven by two vehicle use cases. Where appropriate a comparison with other Commercial-off-the-Shelf (COTS) test automation tools has been considered. The section concludes with a summary and a reflection on benefits and impact to the problem of late detection of failure modes and/or software defects during product development.

The key innovations, benefits, lessons learned and future work resulted from this research are described in Section 6. The claims of innovation are presented together with a summary of key benefits and impact from each of the solutions to the automotive embedded software development. Lesson learned throughout this research are discussed and where appropriate reflections are made.

Concluding remarks are presented in Section 7. Concluding remarks describe how the proposed solutions address the research aim and objectives of this project. Reflection on the production readiness of each solution is discussed.

2. The need to improve automotive embedded software development

This section presents the need to improve embedded software development within the automotive industry. In order to identify areas for potential improvement, the impact of automotive supplier base on embedded software quality was studied followed by a literature review on best practice.

Best practice for automotive embedded software development was captured through literature review in the following areas.

- Automotive development processes.
- Automotive architecture description languages.
- Automotive software and quality standards.
- Model and test exchange standards.
- Test specification methods and tools.
- Approaches for automated generation of test cases and test scripts.

2.1. Impact of automotive supplier base

In order to understand the impact of automotive supplier base on embedded software development two key studies were undertaken. The first study considered the effect from OEM and supplier base integration on embedded software, whereas, the second study assessed the automotive supplier base capability for embedded software development.

2.1.1. Effect from OEM and supplier integration

Due to the embedded software complexity associated with the development of the vehicle, automotive OEMs are adopting supplier integration into their development

processes (Sharma, 2005), (Tang and Qian, 2008). In order to reduce supplier management and co-ordination costs, automotive OEMs are continuously looking for opportunities to reduce the number of direct suppliers. This trend has resulted in a multilevel partnership interface between an automotive OEM and supplier chain as shown in Figure 7.

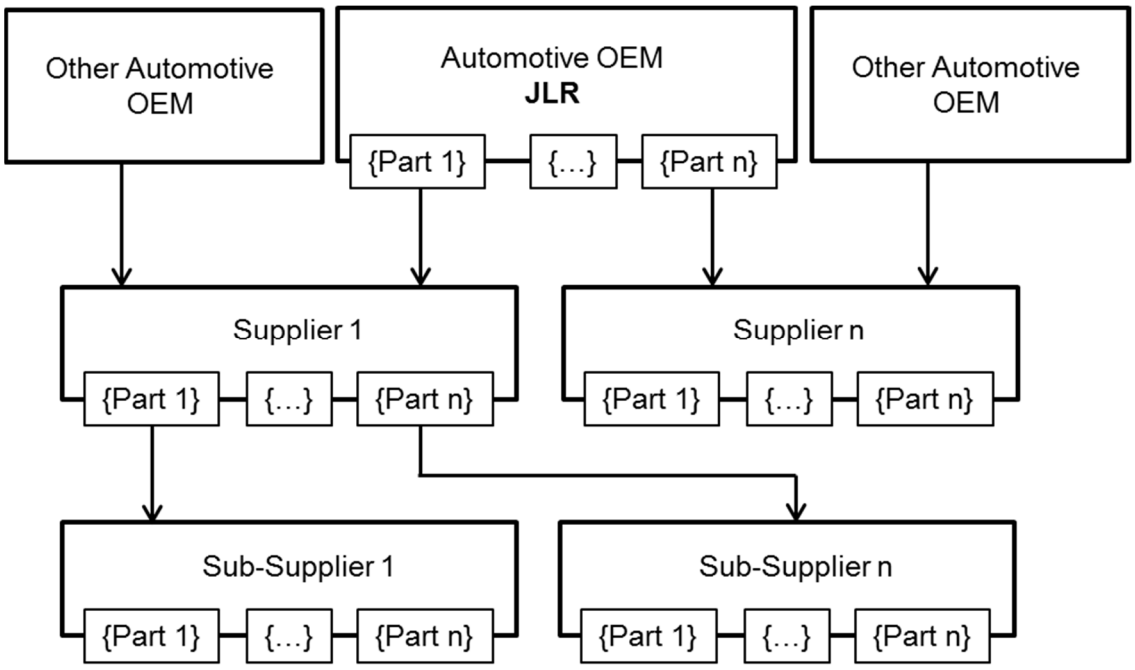


Figure 7. Link between JLR multilevel supplier chain interface and other automotive OEMs.

The most capable suppliers, normally the ones that are responsible for hardware and software integration, are called Tier 1s or system suppliers. Tier 1 suppliers use Tier 2 or sub-suppliers to develop parts of their system (sensors, microprocessors, operating systems, electronic components, etc.). Tier 1 suppliers have direct contact with the OEM and Tier 2 sub-suppliers have direct contact with the Tier 1 suppliers. In this multilevel supplier chain interface the Tier 2 suppliers do not have direct contact with the OEM. Furthermore, it is very common within the automotive industry that OEMs share key Tier 1 and Tier 2 suppliers for the development of their product including

embedded software (Wiengarten *et al.*, 2010), (Thun and Hoenig, 2011). This multilevel supplier chain directly suggests that embedded software development challenges for a company like JLR, driven by its supplier base capability, are also reflected in the development challenges for other OEMs.

A typical automotive OEM embedded software development process involving Tier 1 suppliers is shown in Figure 8.

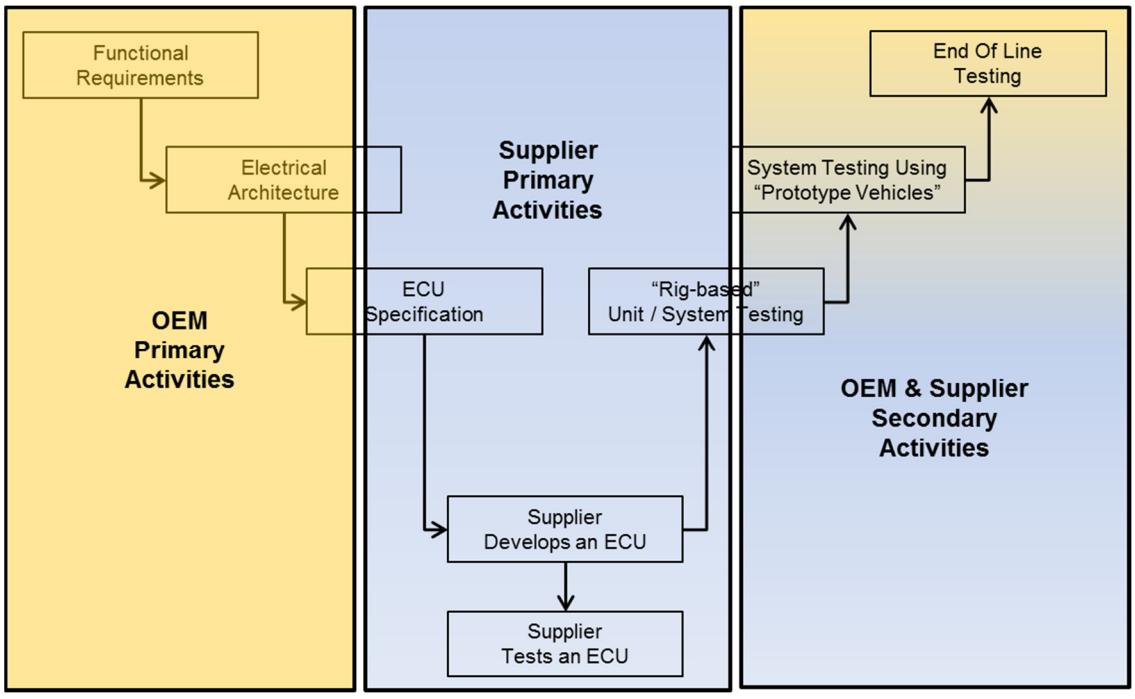


Figure 8. Typical automotive embedded software development process involving a Tier 1 supplier.

The OEM's primary activities are between the definition of the functional requirements and ECU specification. The supplier's primary activities include ECU development and test. The integration phase of the embedded software development includes both OEM and supplier secondary activities. More details regarding the roles and responsibilities for both the OEM and supplier are given in Submission 1. The weaknesses of the embedded software development process shown in Figure 8 are well documented in

the literature (Ebert and Jones, 2009a), (Liggesmeyer and Trapp, 2009a), (Holtmann *et al.*, 2011). In brief these are:

- Lack of proof of concept of the design at the early stages of the development.
- In most cases only a written specification is provided to the supplier for the ECU development.
- Lack of early verification and validation before embedded software development sourcing and commitment is made.
- OEMs must wait until all ECUs are available before integration testing can commence.
- Dependency on physical prototypes and limited use of test automation tools, methods and techniques.

The problem statement resulting from this study is as follows:

An OEM's capability to develop and validate software specifications prior to supplier sourcing, and the supplier task to deliver ECUs with robust embedded software, can have an effect on the capability to detect failure modes and/or software defects during the early stages of the product development process.

2.1.2. Automotive supplier base capability assessment

The effect of automotive OEM and supplier base integration on the embedded software development is undoubtable. The purpose of this sub-section is to assess automotive supplier base capability for embedded software development. As a result a detailed study and analysis was conducted. Full details of this study and analysis are given in Submission 1.

The aim of the study and analysis was:

To collect data from across the automotive supplier base and analyse suppliers' capability for embedded software development.

The main stages of the study and analysis were:

- Development of a questionnaire for data collection.
- Collection and analysis of data.
- Identification of areas for potential improvement.

It is widely accepted within the automotive industry that modelling and computer simulation concepts play a key role in the development and test of automotive embedded software (Zander *et al.*, 2011). According to (Zander *et al.*, 2011) and many other researchers the fundamental elements in modelling and simulation of automotive embedded software are: Model-based Development (MBD); Model-in-the-Loop (MIL); Software-the-Loop (SIL); Rapid Controller Prototyping (RCP); Hardware-in-the-Loop (HIL). All these modelling and simulation concepts can help to detect failures modes and/or software defects during product development (Hoyos Velasco *et al.*, 2012), (Liebezeit and Serway, 2012), (Völter *et al.*, 2013), (Matinnejad *et al.*, 2013), (Osswald *et al.*, 2013). Each modelling and simulation concept is presented in more detail in Submission 2.

A high level structure of the questionnaire is depicted in Figure 9. Its structure includes three main parts; requirements, embedded software development and embedded software test.

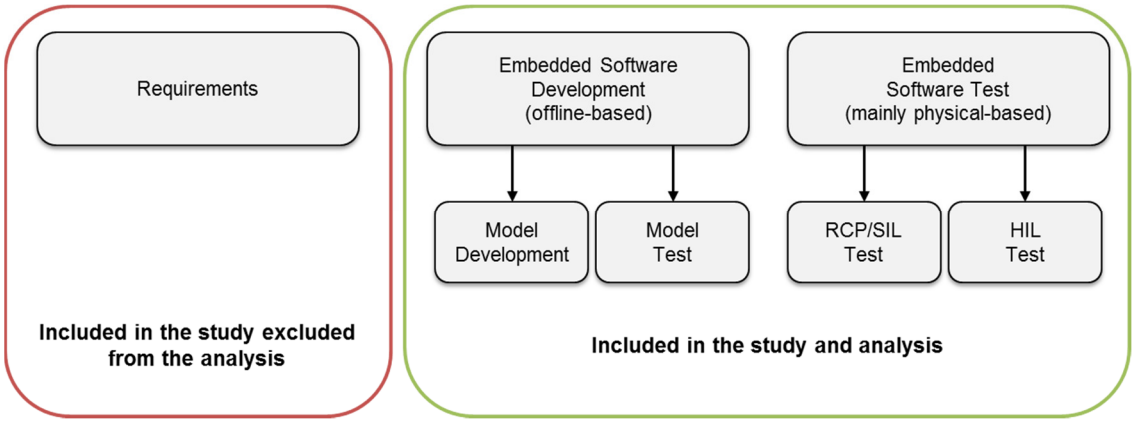


Figure 9. Structure of the questionnaire.

Requirements were included in the questionnaire but were excluded from the capability analysis as it is normally the OEM's responsibility to provide system and software requirements to suppliers. The questions of the first part (embedded software development) were broken down into two sub-groups of model development and model test. The questions of the second part (embedded software test) were broken down into two sub-groups of RCP/SIL test and HIL test. Supplier questionnaire responses were given a capability rating from 1 to 5 as shown in Table 2.

Rating	Meaning
1	None – No capability or no use of a particular concept
2	Low – Very little capability or very little use of a particular concept
3	Moderate – Some capability or some use of a particular concept
4	High – Good capability or good use of a particular concept
5	Very High – Full capability or full use of a particular concept

Table 2. Capability rating given to the supplier questionnaire responses.

It is important to note that the ratings may not accurately represent the actual supplier embedded software capability due to interpretation errors and departmental variation within the individual supplier organisation. In order to minimise the risk of the above,

the questionnaire targeted individuals within suppliers which were responsible for the development and release of embedded software. It should also be noted that the data today may differ since the study was conducted in 2012. The results of the study and analysis are depicted in Figure 10. The data show that:

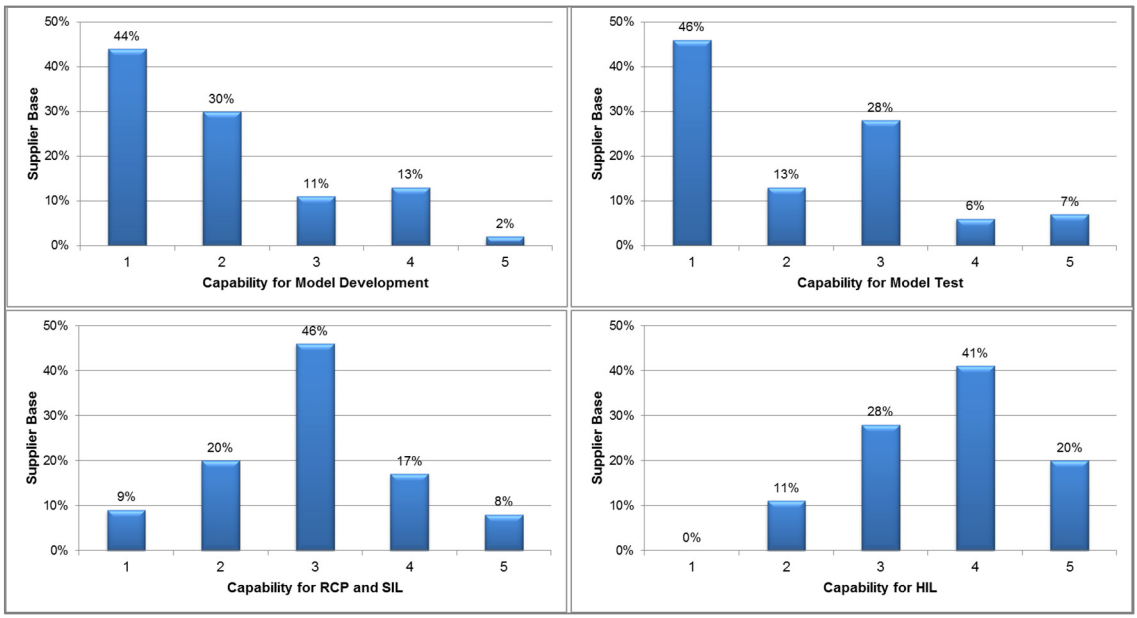


Figure 10. Automotive suppliers' capability for model development, model test, RCP & SIL and HIL.

- The capability or adoption of model development for automotive embedded software is limited. 44% of suppliers have no capability or use model development for embedded software.
- The capability or adoption of model test for automotive embedded software is very low. 46% of suppliers have no capability or use model test concepts for embedded software development.
- The capability or adoption of RCP and SIL for automotive embedded software is moderate. Despite the fact that only 8% of suppliers have full capability or use RCP, SIL concepts for embedded software, 46% of the suppliers have some form of capability or use of RCP, SIL concepts.

- The capability or adoption of HIL for automotive embedded software is high. The data suggest that 61% of the suppliers have good to full capability or use of HIL concepts for automotive embedded software development.

The questionnaire has revealed several challenges associated to automotive software development and test capability across the supplier base. Despite the fact that model-based development and test concepts exists, the majority of the automotive supplier base has limited capability or use of them. The precise root cause of this is not known as every supplier organisation is different. However contributing factors can be traced to a lack of:

- Mature model-based development processes.
- Engineering standards and procedures to allow engineers to work on specific deliverables.
- Engineering skills for model-based development.
- Investment for new tools, methods and techniques.
- Availability of additional resources early in the product development process.
- Understanding of software specification, including requirements engineering coupled with early proof-of-concepts.
- Confidence in new tools, methods and processes as well as organisational mind-set about the importance of embedded software in automotive product development process.

In order to draw further conclusions from this study an average from each sub-group was taken in order to establish if there is a capability trend in all these concepts. Figure 11 shows the average supplier base capability for model development, model test,

RCP/SIL and HIL. It can be concluded that there is a negative trend from HIL concepts to model development. Model development and model test concepts are normally applied early in the development process whereas the use of RCP/SIL and HIL concepts are coupled with the availability of compiled code or production intent ECU. This trend can be linked back to the lack of early failure modes and/or software defects detection profile of a vehicle programme.

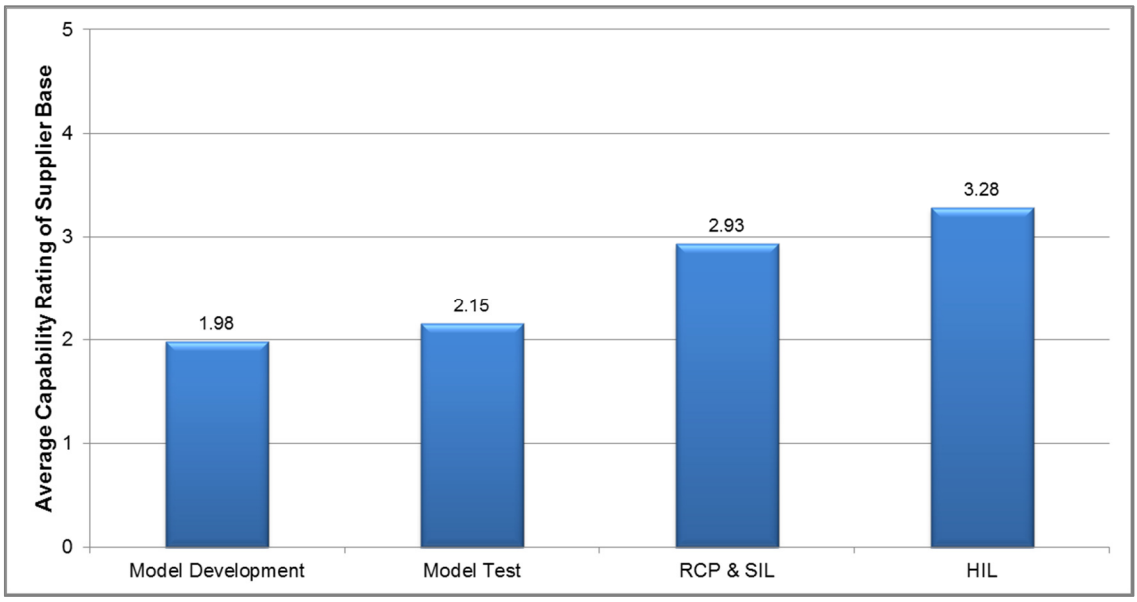


Figure 11. Automotive suppliers' average capability for model development, model test, RCP & SIL and HIL.

The following key observations can be drawn from the supplier capability study:

- Lack of standardised processes across all modelling and simulation concepts within the automotive industry. Without standardised processes in place it is very difficult for any organisation in the automotive industry to build strong engineering skills and invest in new tools, methods and processes.
- Lack of automated testing for automotive embedded software (mainly during the early phases of the development). This finding could suggest that there is an issue

with current best practice in the wider automotive industry in terms of effective generation and use of test data.

- Lack of test data sharing and linking across the different levels of the embedded software development process.

The assumptions, strengths and weaknesses of the approach taken to capture and analyse the supplier base capability for automotive embedded software development and test are given in Submission 1.

The problem statement resulting from this study is as follows:

Data collected and analysed from the automotive supplier base responsible for embedded software development and test suggest that there is a lack of model development and automated testing during the early stages of the development.

The research findings from both studies led to a targeted literature review. The key areas of interest were reviewed as shown in the following sub-sections.

2.2. Automotive development processes

A brief overview of the main automotive development processes reviewed in this research is given in this section. More details on each development process, including figures and diagrams are provided in Submission 2.

The Waterfall product development process is one of the oldest and its origins can be found back in the 1970s (Royce, 1970), (Bell and Thayer, 1976). This development process is used for software development projects based on planning and intensive

documentation. Each phase of the development “pours-over” into the next phase. In the literature there are many variants of the Waterfall model each of them aiming to address shortcomings of this development process (Bassil, 2012), (Patel and Jain, 2013),(Rashid, 2014). The main weakness of the Waterfall model development process is the lack of iterative phases and upfront full commitment to software coding without validated requirements and executable specification. Despite the fact that the Waterfall model is associated with high costs and efforts it is still used today (Petersen *et al.*, 2009). The model offers predictability and focuses on paying attention to planning the architecture of the software in a great detail.

One of the most popular product development processes in the automotive industry is the V-model (Clark, 2009), (Mathur and Malik, 2010), (Oshana and Kraeling, 2013), (Ferreira *et al.*, 2014). It is typically used for software design, validation and testing of a product. In the V-model process the initial design phase typically consists of written specifications of the system, software and hardware design. The next phase involves some form of prototyping (normally hardware prototyping) and then development of the product. Lack of executable specification leads to a great difficulty in keeping specification up to date. Engineers involved at each stage of the development process are subject to strict boundaries between the specification, design, coding, and testing activities. As a result the V-model process is not iterative, hence leading to unnecessary late and expensive changes.

The Product Evolution Process (PEP) typically is used to manage engineering activities involving the design and testing of a product. PEP is also referred to as the time-to-market process. In the automotive industry every OEM has its own detailed process model of PEP. The equivalent of PEP within JLR is the PCS process, mentioned in Section 1. PEP consists of a number of milestones or gateways and phases with

associated set of deliverables. The PEP covers development activities, production activities as well as sales activities. More details on PEP process can be found in (Weber, 2014). The PEP development process is too generic and operates at a high level. Since PEP process is responsible for the whole product (vehicle) development its use to improve embedded software development is very limited.

Agile software development was developed as a simpler and more flexible development process (initially mainly for software) in contrast to the more complex, and structured processes such as the V-model and Waterfall models (Losada *et al.*, 2013), (Eckstein, 2013). Agile thinking is based on short development cycles called “sprints”. The software is developed incrementally and it is a subject to continuous customer reviews that leads to an early integration. Agile development adoption is increasing in the automotive industry as many automotive systems rely on software associated with complex algorithms. Thus, agile development is suitable for small projects rather than large projects as it becomes difficult to judge efforts and the time required for the project in the software development cycle (Balaji and Murugaiyan, 2012), (Eckstein, 2013), (Losada *et al.*, 2013).

Another well know development process is the spiral model for software development (Pressman, 2005). This software development process has four phases: planning; risk analysis; engineering and evaluation. The main phase of the spiral model is the risk analysis phase. A software development project repeatedly passes through the four phases in iterations or so called “spirals”. According to (Munassar and Govardhan, 2010) the main drawback of the spiral model is cost and the engineering expertise to conduct the risk analysis. The main advantage of this development process is the early software development and its suitability for mission-critical systems. In the literature there are several variants of the spiral model but their main principle behind the

process remains the same (Unger and Eppinger, 2011), (Mahmoud and Ahmad, 2013), (Boehm *et al.*, 2014).

According to (Lightsey, 2001) the systems engineering process is a top down iterative process applied sequentially through all stages of the product development. The main engineering activities of systems engineering are requirements analysis, functional analysis and allocation and finally design synthesis. The core idea behind the systems engineering process is to generate an architecture of how the system and its subsystems are linked and integrated (Kossiakoff *et al.*, 2011). Typically, in the automotive industry systems engineering is used to describe the system's interactions at an abstract level whereas the lower level system description and implementation is left to other development processes such as V-model and agile. Systems engineering is well documented in the literature and thus more details can be found in (Axelsson, 2001), (Weilkiens, 2011), (Maurer and Winner, 2013b).

Model-based development (MBD[2]) process has its origins from the world of simulation and made its introduction into the automotive industry in late 1980 (Zander *et al.*, 2011). MBD was first used in automotive industry for safety and powertrain control systems development. Since then research work in the area of powertrain systems development continued strong as shown in recent publications (Hu *et al.*, 2014), (Mallamo *et al.*, 2014), (Peters *et al.*, 2014), (Weibel *et al.*, 2014), (Wu, 2014), (Schick and Paulweber, 2015), (Shukla and PVK, 2015), (Walter *et al.*, 2015). After powertrain, body control systems started to use MDB, mainly for the development of discrete body control functions such as locking, alarm, lighting and vehicle personalisation (Wei *et al.*,

[2] In this research the abbreviation of MBD is used for model-based development. In some publications, the abbreviation of MBD is also used for model-based design. MBD is considered in this research as a broader term that covers model-based requirement engineering, model-based design and model-based testing.

2011), (Schmidt *et al.*, 2014), (Akhtar, 2015), (Föcker *et al.*, 2015a). The aim of the research targeted automated code generation directly from functional or implementation models. Infotainment was the next area within the automotive industry to adopt MBD (Duan *et al.*, 2010), (Ersal *et al.*, 2010), (Huang *et al.*, 2010), (Hess *et al.*, 2012), (Drabek *et al.*, 2013). The research work within the infotainment area mainly focused on model-based requirement and testing of features such as audio, phone and vehicle navigation. A common theme emerging from all the research work in the area of MBD is the need for a set of guidelines that drive which model-based approaches shall be used in the engineering development process of a product. This gap leads to a potential innovation opportunity for integrated methods and processes for automotive embedded software development. The potential of this innovation opportunity is also supported by the work and research findings of (Holtmann *et al.*, 2011), (Liebel *et al.*, 2014), (Marko *et al.*, 2014), (Shahbakhti *et al.*, 2015), (Petrenko *et al.*, 2015a).

2.2.1. Strengths and weaknesses

A summary of the main strengths and weaknesses of the key automotive development processes reviewed in the previous section is shown in Table 3. The key message is:

Existing automotive development processes are too broad to implement and lack standardisation with a clear set of deliverables that engineers can focus on.

Development Process	Strength	Weakness
Waterfall	• Suitable for projects that can be predicted and planned in advanced • More detailed project architecture work at the early phases	• Lack of iterative phases • Lack of executable specification • Increased cost and engineering effort due to late software development and test • Complex and structure process
V-model	• Widely used within the automotive industry and well known to the engineers • Suitable for low complexity product development • Software and hardware liability resigns with suppliers • No need for upfront engineering	• Lack of executable implementation/functional models due to the availability of written specification • Software and hardware design are delivered by suppliers • Late system integration and testing, typically using production hardware and software intent prototypes
Product evolution process (PEP)	• Suitable for managing the whole product development cycle from development to sales	• Only operates at a high level • Requires other development processes to be in place for software development • Lack of specific engineering deliverables
Agile	• Simple and flexible process • Early customer/stakeholder engagement • Deviations from the initial plan are allowed	• Not suitable for large scale projects such as automotive development • Difficult to judge engineering effort and time required to complete the project
Spiral	• Early software development • Iterative process • Suitable for mission critical systems	• Increased costs due to engineering expertise to conduct risk analysis • Increased cost for prototype development at each iteration
Systems engineering	• Suitable for architecture design and development • System interactions can be modelled at the early stages of the development	• Too broad and abstract process • Lack of specific deliverables at each stage of the development • Not suitable for the development of low level software
Model-based development (MBD)	• Early proof-of-concepts through executable specification • Implementation/functional models can be used for auto-coding and offline testing • Good support from Commercial-off-the-Shelf tools	• Significant upfront engineering effort to develop proof-of-concepts and models • Investment in modelling and simulation tools • Engineering effort to develop automated test scripts • Software skills to develop implementation/functional models

Table 3. Main strengths and weaknesses of key automotive development processes.

2.3. Automotive architecture description languages

A brief overview of the main automotive Architecture Description Languages (ADL) reviewed in this research is given in this section. More details on each ADL, including figures and diagrams are provided in Submission 2.

Architecture and Analysis Description Language (AADL) is a modelling language dedicated to embedded systems for the definition of software and hardware

components and their allocation (Feiler and Gluch, 2012). AADL does not provide an appropriate structural framework for automotive systems development as it lacks complementary abstraction levels (Cuenot *et al.*, 2010). AADL is subject to several research efforts where tools and analysis extensions are investigated (Bozzano *et al.*, 2010a), (Bozzano *et al.*, 2010b), (Ma *et al.*, 2013), (Biggs *et al.*, 2014), (Tran *et al.*, 2014).

The Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) was initially developed in the ITEA EAST-EEA project and subsequently refined and aligned with the more recent AUTOSAR automotive standard (MARTE, 2015). Compared to AADL, EAST-ADL has a broader coverage in terms of development life-cycle. It supports hardware, software and infrastructure, as well as other aspects of the development such as requirements, safety and dependability (Qureshi *et al.*, 2014). EAST-ADL relies on AUTOSAR representation for the software architecture and hardware details of the implementation. Current research efforts focus on automatic test-case generation directly from EAST-ADL specification and how internal analysis such as behaviour execution can be transferred back to the EAST-ADL models (Goknil *et al.*, 2014), (Mubeen *et al.*, 2014).

The Modelling and Analysis of Real-Time and Embedded systems (MARTE) is an Object Management Group (OMG) standard for modelling real-time and embedded systems in Unified Modelling Language 2 (UML2), (MARTE, 2015). The MARTE profile architecture consists of three main packages. In the literature there is not strong evidence to suggest that MARTE is ready to be adopted by the automotive industry. MARTE relies on UML adoption, its main strength is the definition of a detailed set of non-functional attributes for enabling scheduling analysis. More details on MARTE profile architecture can be found in (Selic and Gérard, 2013).

Systems Modelling Language (SysML) is the first UML standard for systems engineering proposed by Object Management Group, (OMG, 2015), and initiated as a project in 2003. The main purpose of SysML is to describe functional requirements in graphical or tabular form in order to aid model traceability. Research of how SysML can be used for automotive embedded systems is very active area particularly since 2011, (Góngora *et al.*, 2013a), (Krammer *et al.*, 2013), (Grönninger *et al.*, 2014), (Sporer, 2015). In (Bak *et al.*, 2013), an Adaptive Cruise Control (ACC) module was attempted to be modelled using SysML. The authors concluded that a better tool support is required since SysML and AADL do not deal well with variability. On the other side the authors found that SysML has the potential to be a well-suited modelling language for automotive systems.

2.3.1. Strengths and weaknesses

A summary of the main strengths and weaknesses of key automotive architecture description languages reviewed in the previous section is shown in Table 4. The key message is:

Current automotive architecture description languages are too difficult to implement due to their immaturity and need for specialised engineering skills. Immaturity is linked to the complexity of the existing solutions and a lack of adoption by the automotive industry. Existing automotive architecture description languages require simplification and pragmatic paradigms of how they can be deployed.

Architecture Description Language	Strength	Weakness
Architecture and Analysis Description Language (AADL)	• Suitable for the definition of hardware and software components • Addresses system's interaction and design aspects such as error handling	• No structural framework for automotive systems development due to lack of complementary abstraction levels • No usage within the automotive industry • Requires specialised engineering skills
Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL)	• Multiple abstraction levels provide a multi-viewed and structured information management support of the system • Compared to AADL, EAST-ADL has a broader coverage in terms of development life-cycle	• Relies on AUTOSAR representation for the software architecture and hardware details of the implementation • Lack of automatic test-case generation directly from EAST-ADL specification • Lack of transferability between models and EAST-ADL specification • Requires specialised engineering skills
Modelling and Analysis of Real-Time and Embedded systems (MARTE)	• Standard for modelling real-time and embedded systems • Suitable for the definition of a detailed set of non-functional attributes for enabling scheduling analysis	• Requires knowledge of UML2 • Relies on UML adoption • Not mature to be adopted by the automotive industry
Systems Modelling Language (SysML)	• Enabler for systems engineering • Can be used to describe functional requirements in graphical or tabular form • Can be used without Commercial-off-the-Shelf tools • Adopted by the automotive industry	• Not suitable for implementation/functional model development • SysML graphical views can become unmanageable for large scale projects • Inconsistent use of SysML structural and behavioural views

Table 4. Main strengths and weaknesses of key automotive architecture description languages.

2.4. Automotive software and quality standards

A brief overview of the main software and quality standards used within the automotive industry is given in this section. More details including figures and diagrams are provided in Submission 2.

The Automotive Open System Architecture (AUTOSAR) consortium introduced in 2004 a new concept for automotive embedded software development (AUTOSAR, 2015). The aim of the concept was to decouple the application software from the ECU

middleware and hardware. AUTOSAR has penetrated the automotive industry and it is now considered as the core standard for application software development (Lee *et al.*, 2013), (Briciu *et al.*, 2013), (Reinhardt *et al.*, 2013), (Wang *et al.*, 2014). AUTOSAR focuses on specific ECU application software development and does not address interaction and interface issues across networked components and systems.

The maturity of an organisation's software development process can be measured using the Capability Maturity Model Integration (CMMI) process (Chrissis *et al.*, 2003). The process is consisting of five levels ranging from a process considered to be unpredictable to a process which is controlled and its focus is on improvement. CCMI requires an established software development process to be in place. As a result OEMs still need to decide and chose which software development process to adopt.

Similar to CMMI, Automotive Software Process Improvement and Capability dEtermination (Automotive SPICE) is a framework for designing and assessing software development processes. Automotive SPICE has been developed under the Automotive SPICE initiative by consensus of some vehicle manufacturers within the automotive Special Interest Group (SIG) (Automotive SPICE, 2015). The aim of the Automotive SPICE is to become an international standard in the automotive industry for better processes and better product quality. A number of research activities can be found in the literature (mainly since 2010) aiming to address issues related to Automotive SPICE integration with other standards, tools, methods and processes (Petry, 2010), (Messnarz *et al.*, 2011), (Steiner *et al.*, 2012), (Mellegard *et al.*, 2012), (Gallina *et al.*, 2014), (Lami and Falcini, 2014). Neither CMMI nor Automotive SPICE specifies how defects documentation and analysis is to be done.

In addition to CMMI and Automotive SPICE the automotive industry has started to implement the ISO 26262 functional safety standard (ISO 26262, 2011). Written specifically for the automotive industry, the standard is adapted for the V-model of product development and consists of nine parts. In the literature, significant amount of research work takes place in order to address the challenge of how best to implement and integrate ISO 26262 functional safety standard with existing product development processes, methods and techniques (Beckers *et al.*, 2013), (Rana *et al.*, 2013), (Lee *et al.*, 2014), (Krieg *et al.*, 2013), (Khabbaz Saberi *et al.*, 2015). In addition to that ISO 26262 has limited coverage for product development processes that are driven by model-based development.

2.4.1. Strengths and weaknesses

A summary of the main strengths and weaknesses of key automotive software and quality standards reviewed in the previous section is shown in Table 5. The key message is:

Automotive quality standards are only suitable for assessing existing software development processes. Automotive software standards are either too focused on software implementation and software re-use at ECU level or too broad covering the whole product development cycle for system development areas such as functional safety.

Software and Quality Standard	Strength	Weakness
Automotive Open System Architecture (AUTOSAR)	• Decouples the application software from the ECU middleware and hardware • Software re-use across different ECUs due to standardisation of the application software • Adopted by the automotive industry	• Does not address interaction and interface issues across networked components and systems • Requires significant amount of on-board processing power • Cannot be used as a software development process on its own
Capability Maturity Model Integration (CMMI)	• Can be used to measure the maturity of a given software development process • Can drive improvements into an existing software development process	• Requires a software development process to be in place • Incremental resource is required to manage audits and drive improvements • Lack of defect documentation and analysis
Automotive Software Process Improvement and Capability dEtermination (Automotive SPICE)	• Can be used as a framework to design and assess software development processes • Driven by the automotive industry •	• Requires a software development process to be in place • Not an international standard • Incremental resource is required to manage audits and drive improvements • Lack of defect documentation and analysis
ISO 26262 functional safety standard for road vehicles	• International standard for automotive functional safety • Written specifically for the automotive industry • Adapted to V-model development process • Provides complementary guidelines for software development best practice	• Limited coverage for product development processes that are driven by model-based development • Requires a software development process to be in place

Table 5. Main strengths and weaknesses of key automotive software and quality standards.

2.5. Model and test exchange standards

A brief overview of the main model and test exchange standards used within the automotive industry is given in this section. More details including figures and diagrams are provided in Submission 3.

The Functional Mock-up Interface (FMI) is claimed to be a tool independent standard for the exchange of simulation models (FMI, 2015). Its aim is to improve the exchange of simulation models between suppliers and OEMs. Over the past couple of years

(mainly since 2013), industry and science have shown an interest in FMI and its use to allow model exchange and co-simulation (Blochwitz *et al.*, 2011), (Blochwitz *et al.*, 2012), (Awais *et al.*, 2013), (Exel *et al.*, 2014). FMI will require a number of changes and additions in order to satisfy all the requirements of the automotive suppliers and OEMs.

The Open Diagnostic data eXchange (ODX) is a new industry standard aimed specifically for ECU diagnostics development and test (BSI ISO 22901, 2011). ODX development has been driven from the need for standardised vehicle electronic description formats for diagnostics and flash programming data (Yun and Lee, 2011), (Li and Tang, 2013), (Natterer *et al.*, 2014).

Traditionally the automotive industry mainly relied on paper documentation and/or proprietary authoring environments to document and to implement diagnostic test sequences. The effort to create these diagnostics test sequences can be reduced with the adoption of Open Test sequence eXchange (OTX) standard (BSI ISO 13209, 2011). OTX is a new standard which is open and standardised with the purpose to provide a machine readable description of diagnostic test sequences for ECU diagnostics development and test. OTX has started to penetrate the automotive industry since 2013 (Aichernig *et al.*, 2012), (Subke, 2014), (Subke and Eberl, 2015). The combination of ODX and OTX provides an open and standardised process for ECU diagnostics development and test but does not address the validation and verification of vehicle features and systems driven by embedded software development.

The Automotive Test eXchange (ATX) standard (not an international ISO standard) is an initiative of the Association for Standardisation of Automation and Measuring

Systems (ASAM), (ASAM, 2015). ASAM is a membership only association and its main purpose is to coordinate the development of technical standards, which are developed by working groups composed of experts from member companies mainly automotive OEMs. The lack of standardised test exchange between automotive OEMs and suppliers has led the ASAM association to work on a new initiative called ATX. ATX promises to cover test activities such as test specification, test planning, test execution and test evaluation. ATX is at the early stages of the development (first release for ASAM members only in 2013) and as a result there is no evidence in the literature of a single adoption within the automotive industry (literature search up to 2015 has found no publications of conference or journal papers).

The recent initiative of the main automotive OEMs to work towards the creation of a standardised test exchange confirms that this is not a JLR specific issue or challenge but a wider automotive industry one.

2.5.1. Strengths and weaknesses

A summary of the main strengths and weaknesses of key automotive model and test exchange standards reviewed in the previous section is shown in Table 6. The key message is:

Automotive model and test exchange standards do not address functional testing of embedded software. In addition to that they are either not ready for implementation or mainly focus on vehicle diagnostics and exchange of simulation models. The lack of readiness is linked to standards not being available through international standardisation and the need for further testing to prove their effectiveness on automotive embedded software.

Model and Test Exchange Standard	Strength	Weakness
Functional Mock-up Interface (FMI)	• Tool independent standard for the exchange of simulation models between suppliers and OEMs • Since 2010 more than 30 simulation packages/tools support FMI	• Lack of data type support mainly for code optimisation. • Parameters, states, inputs, and outputs of the model are not exposed directly to the outside world • Little evidence for automotive industry adoption
Open Diagnostic data eXchange (ODX)	• Tool independent industry standard aimed specifically for ECU diagnostics • Adopted by the automotive industry • Driven by UML data specification and XML implementation format	• Not applicable for functional software development
Open Test sequence eXchange (OTX)	• Industry standard for diagnostic test sequences • Adopted by the automotive industry • Driven by UML specification for test sequences and XML implementation format	• Not applicable for functional software development • Not suitable for functional and non-functional based testing
Automotive Test eXchange (ATX) standard	• XML description of test cases • Has the potential to cover test activities such as test specification, test planning, test execution and test evaluation	• Not an international standard • At the early stages of its development • No evidence in the literature of a single adoption within the automotive industry • No visibility outside the ASAM association • Requires the integration of other ASAM standards for test specification and test execution particularly for HIL systems • Lack of control flow

Table 6. Main strengths and weaknesses of key automotive model and test exchange standards.

2.6. Test specification methods and tools

The process of automotive embedded software development becomes more complicated with the raising complexity of software verification and validation (V&V) requirements. V&V requirements (also called design verification methods or test specifications) are required in order to sign off the final product in terms of:

"Are we building the product right?" – ***Verification***

"Are we building the right product?" – ***Validation***

Validation checks whether the product meets the customer's needs. Verification checks whether the product is well-engineered and free of defects/errors. Verification determines whether the product is robust and of high quality, but it will not ensure that the product is useful to the customer (Hull *et al.*, 2010), (Quirk, 2012).

The need to develop V&V requirements which are correct, precise, consistent, unambiguous, verifiable and complete have attracted the interest of many researchers over the past 5 years (Pohl, 2010), (Alagar and Periyasamy, 2011), (Laplante, 2013), (Pahl and Beitz, 2013), (Wiegers and Beatty, 2013), (Achimugu *et al.*, 2014), (Gigante *et al.*, 2015). In the literature software verification and validation (V&V) requirements are expressed as informal, semi-formal and formal (Cooper *et al.*, 2014). The remainder of this sub-section presents related research and theory in the area of informal, semi-formal and formal specifications methods. More details can be found in Submission 4.

Informal methods for software verification and validation (V&V) requirements are based on natural language (Insfran *et al.*, 2014). Requirements written in natural language are very often lengthy and complex. The main drawback of expressing V&V requirements in natural language is that it is ambiguous and open to interpretation. Despite this drawback, informal methods are still used throughout the automotive industry for the specification of embedded software V&V requirements (Daun *et al.*, 2015). Tools such as Quality Analyser of Requirements Specification (QuARS) were developed for automatic quality evaluation of natural language requirements (Fabbrini *et al.*, 2001), (Lami *et al.*, 2004), (Bucchiarone *et al.*, 2005).

Semi-formal methods for software verification and validation (V&V) requirements are based on a mixture of mathematical formulae (including graphical representations) and natural language. The combination of mathematical formulae and natural language provides a bridge between informal and formal methods hence appealing to the intuition of the user (Di Guglielmo *et al.*, 2010). Unified Modelling Language (UML) is the most common semi-formal approach used in the industry (McUmber and Cheng, 2001), (Bernardi *et al.*, 2013), (Levendovszky *et al.*, 2014). The main advantage of UML modelling language is its potential to transfer semi-formal requirements into formal methods representation (Zafar and Alhumaidan, 2011). User Requirements Notation (URN) is another semi-formal language used to capture both functional and non-functional requirements (Sikandar-gani, 2003), (Saleh and Al-Zarouni, 2004), (Mussbacher *et al.*, 2013). The literature shows no evidence to suggest URN usage within the automotive industry. Entity Relationship Diagrams (ERD) are also used for the definition of semi-formal requirements (Song *et al.*, 1995), (Waugh *et al.*, 2004). ERD are not popular within the automotive industry due to a number of constraints, such as lack of representation of data manipulation. Structured Analysis (SA) is a widely used semi-formal methodology (Ross, 1977). SA is considered to be a strong semi-formal methodology for representing the data processing parts of a system. SA is not suitable though for reactive systems with control structures (Shoval, 1988), (Semmens *et al.*, 1992). Petri Nets (PN) are another approach to semi-formal specification (Molloy, 1982). PNs are used for modelling distributed systems but have limited usage in the automotive industry as they tend to become very large for substantially complex system interactions. State Transition Diagrams (STD) are similar to Petri Nets in terms of strengths and weaknesses but simpler and easier to understand (Grosu *et al.*, 1996), (Olivé, 2007). STD are widely used in the automotive industry for the development of complex control algorithms such as comfort and climate control systems (Kakade *et al.*, 2010), (Wang *et al.*, 2010), (ter Beek *et al.*, 2011), (Liu

et al., 2013a), (Petrenko *et al.*, 2015b). Finally, Data Flow Diagrams (DFD) are another semi-formal graphical technique that is used to represent flow of information (Larsen *et al.*, 1994). Similar to Petri Nets, data flow diagrams have limited usage in the automotive industry as they tend to become very large for substantially complex system interactions.

Formal methods for V&V requirements are based on a symbolic language having a precise mathematical logic and syntax (Boca *et al.*, 2010). Formal methods create precise and unambiguous interpretation of what has been specified. On the other hand formalising specification for large scale systems can be a challenge as it is difficult to describe mathematically software interfaces across many systems. The complexity of mathematics involved requires significant effort for training software developers and test engineers (Liu *et al.*, 2011). The most common formal methods language is the Z notation (Woodcock and Davies, 1996). Z notation is based on set theory and mathematical logic. Z is not used within the automotive industry mainly due to the fact that it does not describe non-functional requirements such as usability and performance. Z has more usage within the aerospace industry, telecommunication industry and universities (Gerhart *et al.*, 2012), (Hinchey and Bowen, 2012). Another formal methods specification language is The Vienna Development Method (VDM). VDM is based on discrete mathematics including set theory. Similar to Z specifications, VDM specifications are normally not machine executable. There is no indication in the literature of VDM usage within the automotive industry. The third most commonly used formal method is B notation (Lano and Haughton, 1996). Similar to Z, the B formal method is based on predicate logic and set theory. There are more formal methods and languages in the literature, however the description of these are outside the scope of this research. More details can be found in Submission 4.

2.6.1. Strengths and weaknesses

A summary of the main strengths and weaknesses of key specification approaches reviewed in the previous section is shown in Table 7. The key message is:

Informal specification approaches can be ambiguous and open to interpretation and formal specification approaches are too complex to implement. There is a potential in semi-formal specification approaches, however, in their existing form they cannot be used for the definition of vehicle test specifications (test cases). They lack intuitive user interfaces that can provide engineers with a specification environment that is easy to understand without the need of specialised skills being present.

Approach	Method	Strength	Weakness
Informal	Natural Language	• Easy to use • No training is required • Easy to communicate with key stakeholders • Complex systems can be described in a relative short period of time	• Can be ambiguous and open to interpretation • Can be lengthy and complex
Semi-Formal	Data Flow Diagrams	• Easy to understand and learn • Easy to depict the flow of information	• Time consuming and difficult to read and translate for large scale systems • Different symbols and models exist
	Entity Relationship Diagrams	• Easy to understand and learn	• No representation of data manipulation
	State Transition Diagrams	• Graphical representation of discrete systems • Easy to communicate events, state transitions, messages and actions • Suitable for automotive systems	• All states of the system must be defined hence difficult for large scale systems
	Structure Analysis	• Suitable for different levels of abstraction • Encourages a rigorous understanding of the requirements	• Not easy to maintain models due to design structure • Not precise semantics can cause ambiguities
	Unified Modelling Language	• Suitable for modelling object oriented systems • Can be used for complex systems • Standardised syntax and semantics	• Tool support required for large scale systems
	User Requirements Notation	• Specifies the relationships between non-functional and functional requirements	• Not suitable for real-time control loops and logic
	Petri Nets	• Suitable for concurrent and parallel systems modelling • Suitable for communication protocols	• Not suitable for large scale systems as they become less readable
Formal	Z	• Suitable for model oriented specification style	• No executable specification • Requires high level of technical skill
	B	• Suitable for model, sequential and property oriented specification style • Executable specification	• Requires high level of technical skill
	VDM	• Suitable for model and process oriented specification style • Suitable for the development of compilers and databases	• No executable specification • Requires high level of technical skill

Table 7. Main strengths and weaknesses of key specification approaches.

2.7. Approaches for automated generation of test cases and test scripts

The cost of testing automotive embedded software contributes significantly to the overall vehicle development cost. Automotive OEMs are continuously seeking new methods and processes in order to reduce the cost of testing and the time it takes to create test cases and execute test scripts. The development of test scripts is not required if test cases are to be executed manually using a prototype vehicle or ECU. Test scripts are only required if they are to be executed automatically. The engineering effort of automated execution of test scripts can be significantly reduced if test scripts are auto-generated from test specifications.

Model-driven development and model-based testing are well-known approaches for test case generation in the field of automotive engineering (Javed *et al.*, 2007), (Schieferdecker, 2012), (Marinescu *et al.*, 2014), (Marinescu, 2014). With model-based testing, models derived from requirements form the mechanism to auto-generate test cases and test scripts. The weakness of this approach is that very often representative functional models of the system under test do not exist. However, cases of automatic generation of system test cases from use case specifications and MATLAB/SIMULINK models can be found in the literature (He *et al.*, 2011), (Mohalik *et al.*, 2014), (Wang *et al.*, 2015). Another approach for automated test case generation is the TestML (Grossmann *et al.*, 2006), (Grossmann and Müller, 2006). TestML is a test description language, which was developed for the interchange of test descriptions. TestML requires specific engineering skills. In addition to that TestML is not mature as it is still reported in the literature as being under review and testing. No evidence in the literature of its use within the automotive industry. An emerging standard from the European Telecommunications Standards Institute (ETSI) is the Test Description Language (TDL). In (Boulet *et al.*, 2015), can be found that for effective test case

generation using TDL, post-processing of test scenarios is required including when and how to introduce concrete data in the generation of executable test cases. There are more less known approaches in the literature. More details for these can be found in Submission 4.

2.7.1. Strengths and weaknesses

A summary of the main strengths and weaknesses of key approaches for automated generation of test cases and test scripts reviewed in the previous section is shown in Table 8. The key message is:

Existing approaches for automated generation of test cases and test scripts are difficult to implement as they require either a model of the system under consideration and/or specialised set of engineering skills.

Approaches for automated generation of test cases and test scripts	Strength	Weakness
Model-based Testing (MBT)	• If a functional model representation of the requirements exist then can be used to automatically generate test cases	• Require high fidelity model of the requirements • Pass and fail criteria must be modelled for automated generation of test scripts
TestML	• Covers test descriptions at different levels of abstraction and is independent from the respective tool environment	• Requires Required specialised skills • Not yet available for industrial practice
Test Description Language (TDL)	• Suitable for covering abstract description of tests for communicating systems	• Requires requirements to be modelled • No use within the automotive industry
Other approaches such as Z, Controlled Natural Language (CNL), Software Cost Reduction (SCR), Cuckoo Search (CS), Tabu Search (TS)	• If mathematical or a functional model representation of the requirements exist then can be used to automatically generate test cases	• Require either mathematical or functional model representation of the requirements • Required specialised skills • Do not cover auto-generation of test scripts • No use within the automotive industry

Table 8. Main strengths and weaknesses of key approaches for automated generation of test cases and test scripts.

2.8. Areas for potential improvement

The findings from the automotive supplier base study and literature review on automotive embedded software development related topics has identified a number of areas for potential improvement. These areas are:

- Standardisation of automotive development processes. Lack of process standardisation results in existing modelling and simulation concepts such as model-based design and model-based testing are not being utilised. Development processes must be pragmatic and easy to understand and deploy within the automotive organisations.
- Automated testing at the early stages of the product development. The findings from the supplier base capability assessment clearly indicate the need to improve the area of early verification and validation.
- Creation of test exchange standards. Current test exchange standards are not suitable for embedded software functional testing or are not matured for the industry to adopt them. An improvement in this area will enable more efficient and effective test resources utilisation.
- Test case creation and automated test script generation. An improved test specification method for test case creation and automated generation of test scripts will enable better testing of embedded software across all different levels of system abstraction.

3. Research methodology

A systematic research methodology has been adopted for this project in order to achieve the research aim and objectives presented in the previous section.

The research methodology of this research project is depicted in Figure 12. The research methodology consists of 4 phases. Each phase involves specific research and development activities aligned to the aim and objectives of the project. A description of each phase of the research methodology is given below.

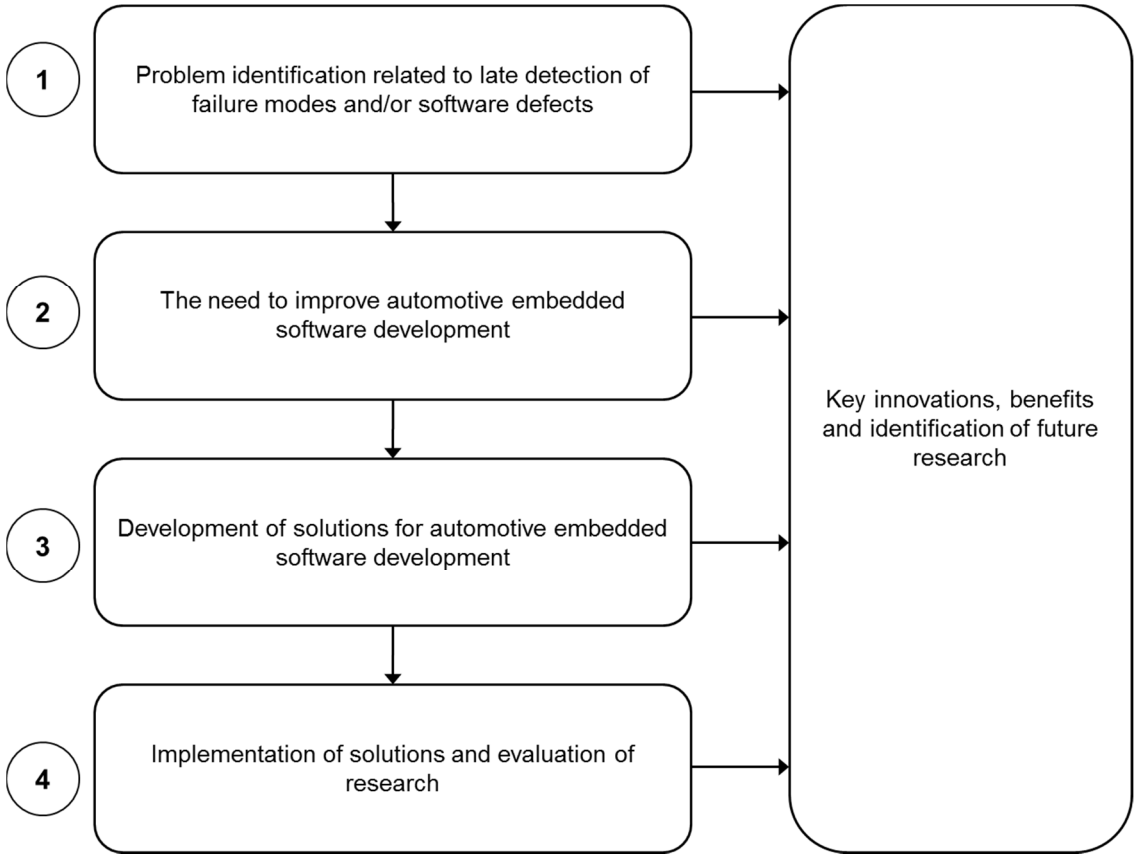


Figure 12. Research methodology.

- **Phase 1 – Problem identification**

  The first phase of the research methodology was focused on the problem identification. The problem identification for this research project related to automotive embedded software development was examined. The investigation and analysis involved a real world case study derived from a Jaguar Land Rover (JLR) vehicle programme development. The focus of the study was on the detection of failure modes and/or software defects during a vehicle programme development process. The emphasis of the analysis was centred on the detection points during the product development.

- **Phase 2 – Best practice and need to improve automotive embedded software**

  The second phase of the research methodology was concentrated on the literature review for best practice in the area of automotive embedded software development. Best practice literature review was driven from a supplier base capability analysis. The analysis was focused on embedded software development and utilisation of existing modelling and simulation concepts. The research findings of the literature review and supplier capability analysis led to the identification of a number of areas for potential improvement. The literature review aimed to provide expert knowledge and understanding in the following engineering areas:

  > Model-based approaches for automotive embedded software development.
  > Automotive product development processes.
  > Automotive architecture description languages
  > Automotive software and quality standards.
  > Model and test exchange standards.
  > Test specification methods and tools.

> Approaches for automated generation of test cases and test scripts.

- **Phase 3 – Development of new and innovative solutions**

  The research outcome from the literature review together with the problem and the identified areas for potential improvement drove the research and development of four solutions. The solutions were aimed to shift failure modes and/or software defects detection to the early stages of the product development process. A set of design requirements for these solutions were established in order to commence the development. Design requirements were later used for the evaluation of the proposed solutions. The solutions developed are addressing the following research priorities:

  > Development of a standardised automotive development process that it is capable to utilise existing modelling and simulation concepts such as model-based design and model-based testing.
  > Development of a standardised test exchange interface in order to enable more efficient and effective test resources utilisation.
  > Development of a test specification method for test case creation and automated generation of test scripts.
  > Development of a holistic end-to-end solution to support automation across all levels of system abstraction with reduced engineering effort and time.

- **Phase 4 – Evaluation of research**

  The fourth phase of the research methodology was focused on the evaluation of research. A set of evaluation criteria, based on initial design requirements, were used to evaluate all proposed solutions for automotive embedded software

development. The evaluation of research included the deployment of the solutions within JLR where appropriate. Typical vehicle case studies were used in order to evaluate the effectiveness of the proposed solutions. Evaluation results were analysed and where appropriate comparison studies were conducted against other Commercial-off-the-Shelf (COTS) tools.

The research outcome from each phase provided the opportunity for feedback and further improvement.

4. Development of innovative solutions for automotive embedded software

This section presents the creation and integration of four solutions suitable for automotive embedded software development. These solutions are:

- Model-based Product Engineering (MBPE) process
- Generic Design Verification Interface (DVI)
- Standardised Design Verification Method (SDVM)
- Platform Independent Test System (PITS)

4.1. Model-based product engineering process

The literature review on automotive development processes concluded that existing processes are too broad to implement as they lack standardisation with a clear set of deliverables that engineers can focus on. An improved automotive development process will enable better utilisation of existing modelling and simulation concepts and as a result has the potential to shift failure modes and/or software defects detection to the early stages of the product development process. The development of the proposed Model-based Product Engineering (MBPE) process was based on the concept of Model-based Development (MBD). An overview of a typical development process containing model-based and non-model-based activities is shown in Figure 13. More details about the development activities depicted in Figure 13 and the motivation for adopting MBD concepts are given in Submission 2. Core model-based activities such as abstract model development (Andrianarison and Piques, 2010), (Apvrille and Becoulet, 2012), (Gorschek *et al.*, 2012), (Góngora *et al.*, 2013b), (Braun *et al.*, 2014b), (Insfran *et al.*, 2014), (Daun *et al.*, 2015), (Mjeda and Hinchey, 2015), functional (or implementation) model development (Schnabler and Stifter, 2014), (Stauder *et al.*,

2014), (Yu *et al.*, 2014), (Föcker *et al.*, 2015b), (Shahbakhti *et al.*, 2015), (Yu *et al.*, 2015) as well as automated generation of production code (Currie *et al.*, 2012), (Inagaki, 2013), (Liu *et al.*, 2013b), (Walters *et al.*, 2014), (Riid *et al.*, 2015) have gained substantial momentum in the recent years. This trend is followed by the use of model-based testing activities such as Model-in-the-Loop (MIL) and Hardware-in-the-Loop (HIL) automated testing (Vora *et al.*, 2014). More details on these generic model-based testing concepts are given in Submission 2 and a graphical representation of how these were used in this research is given in Appendix B.1 to B.4.

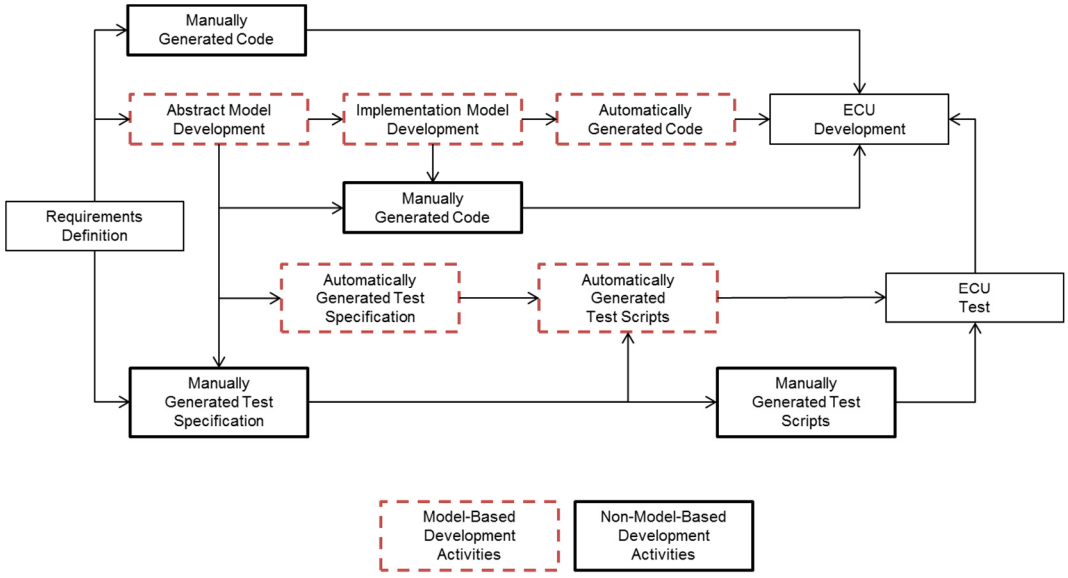


Figure 13. Overview of a typical development process consisting of model-based and non-model-based activities.

4.1.1. Requirements definition for the proposed process

A set of requirements, shown in Table 9, were defined for the development of the proposed Model-based Product Engineering (MBPE) process. The aim of these requirements was to drive the development of a process that addresses the weaknesses and gaps of existing development processes. These requirements are explained in more detail in Submission 2. The core focus of the requirements was on

the process standardisation and clear definition of engineering activities aiming at the early stages of development. A number of initial iterations took place, involving internal JLR stakeholders such as test engineers, in order to review and agree the final set of requirements and their associated pass criteria.

Number	Requirement
1	Standardised
2	Tool independent
3	Link and integration to JLR PCS
4	Support of offline (PC) model-based automated testing
5	Support of real-time model-based automated testing
6	Support of functional/non-functional and vehicle diagnostics testing
7	Design verification methods abstraction to support different vehicle levels
8	Support of test data exchange
9	Support of product complexity
10	Support of agile and concurrent engineering
11	Re-use of executable functional models via model-based design
12	Support of quality, safety and software related industry standards
13	Automated generation and re-use of test scripts via common design verification interface
14	Software verification and validation prior to supplier release
15	Support vehicle level model integration prior to production software and hardware
16	Support customer and system requirements validation
17	Traceability between different levels of abstraction and models
18	Support of OEM/Supplier information management
19	Integrated deployment strategy for robust introduction within a large organisation

Table 9. Requirements for the model-based product engineering process.

4.1.2. Overview of the proposed process

An overview of the proposed Model-based Product Engineering (MBPE) process is shown in Figure 14. The MBPE process consists of six levels of development and test engineering activities. The first three levels are mainly applicable to model-based embedded software development whereas the last three levels are applicable to all software development approaches including model-based and non-model-based. The

process is iterative rather than a waterfall as it allows engineers to move amongst different levels if the necessary entry conditions at each level are satisfied. In addition to that the process is agile as vehicle features and systems can be developed independently. The process begins with the Voice of the Customer (VoC). Typically the VoC is driven by product marketing, market research, engineering research and vehicle attributes departments. The process ends when all vehicle systems are integrated and are at the manufacturing stage for an end of line testing. Three levels of development, verification and validation are executed prior to the ECU development or supplier nomination. Once the first ECU becomes available, the remaining three levels focus on verification and validation of the integrated embedded software consisting of production intent hardware and low level code. Each level is driven by a specific set of engineering activities aimed to be completed at a certain vehicle programme gateways.

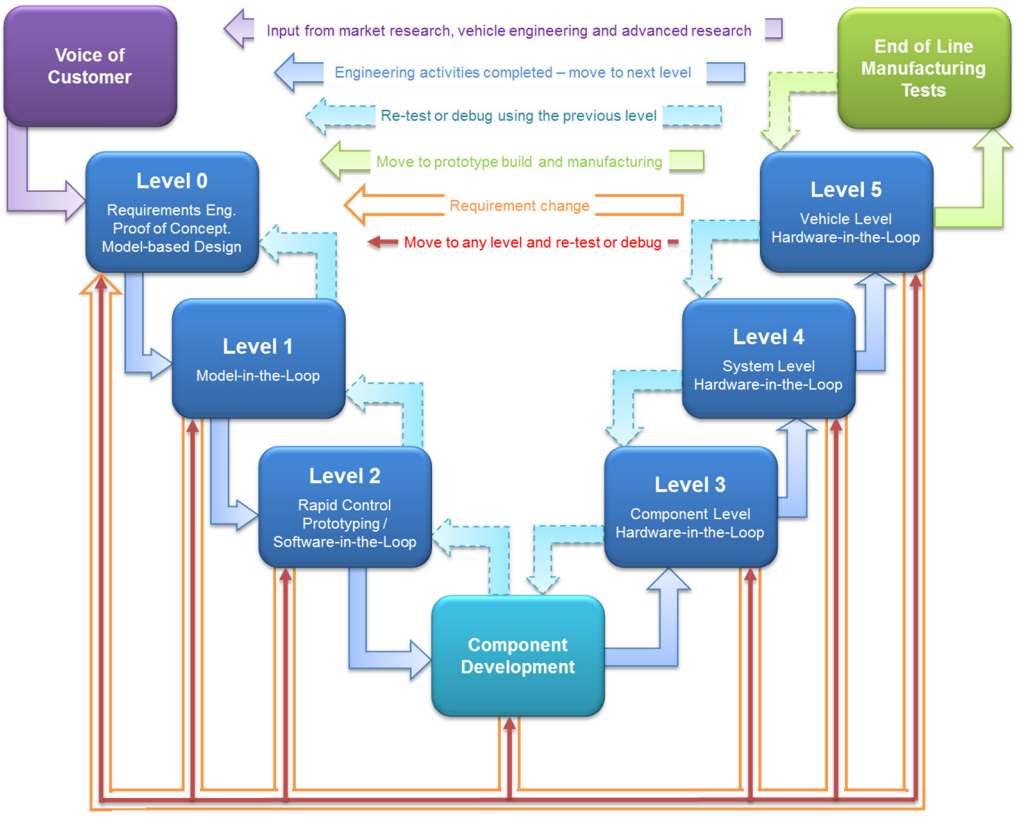


Figure 14. Overview of the proposed model-based product engineering process.

A high level summary of each level of the proposed MBPE process is given as follows:

- Level 0 – Requirements Engineering (RE), Proof of Concept (PoC) and Model-Based Development (MBD): This level describes the engineering activities for the analysis of requirements and creation of functional models.
- Level 1 – Model-in-the-Loop (MIL): This level describes the verification and validation engineering activities for producing functional models suitable for auto-coding.
- Level 2 – Rapid Controller Prototyping (RCP)/Software-in-the-Loop (SIL): This level describes the engineering activities for real-time and offline automated testing. Automated tests are executed directly on functional models or on the auto-generated code. Validated functional models are used for the production code generation and integration with other embedded software and hardware during the component development phase.
- Level 3 – Component level Hardware-in-the-Loop (HIL): This level describes the engineering activities to be followed in order to test the embedded software within the Electronic Control Unit (ECU) or component on its own, in a HIL environment.
- Level 4 – System level HIL: This level describes the engineering activities to be followed in order to validate vehicle feature functionality at a system level by integrating all ECUs attributed to this feature in a HIL environment.
- Level 5 – Vehicle level HIL: This level describes the engineering activities to be followed in order to validate vehicle features in a full vehicle HIL environment. The Level 5 HIL validation is followed by an end of line testing which is carried out at the manufacturing facilities.

The details of each level are given in Submission 2.

4.1.3. Sample of a Level

Event-driven Process Chain (EPC) diagrams were used to depict the engineering activities of the Model-based Product Engineering (MBPE) process. EPC is a flowchart based diagram that can be used for resource planning and identification of potential process improvements (Keller *et al.*, 1992), (Mendling *et al.*, 2005), (Kindler, 2006). More details on EPC are given in Submission 2. The EPC symbols used to describe the MBPE process are given in Table 27 in Appendix A.

Level 1 is chosen as a sample in order to illustrate how the proposed process is used. Level 1 is subdivided into two engineering phases as shown in Figure 15. During the first phase models are verified for compliance with agreed design standards and naming conventions and as a result a compliance verification report is generated. In addition to verifying the design style and naming conventions, models are checked for compliance with other industry architecture standards such as AUTOSAR, and functional safety standards such as ISO 26262. The compliance report details any deviation identified with respect to these industry standards along with reasons for each deviation. A Graphical User Interface (GUI) is prepared to provide a means of feeding input values to the model and observe the values of all the output interfaces. Model validation is carried out using the GUI framework for the tests defined in the model test method. Testing using the GUI framework ensures that the entire model input and output interfaces are verified and their behaviour is as specified in the Software Requirement Specification (SRS) and model Input Output (IO) interface documents. Test scenarios specified in the model test method are used to form the automated test cases. Automated testing is performed to prove the functional behaviour and robustness of the model. Test reports are generated for all tests conducted at each model release.

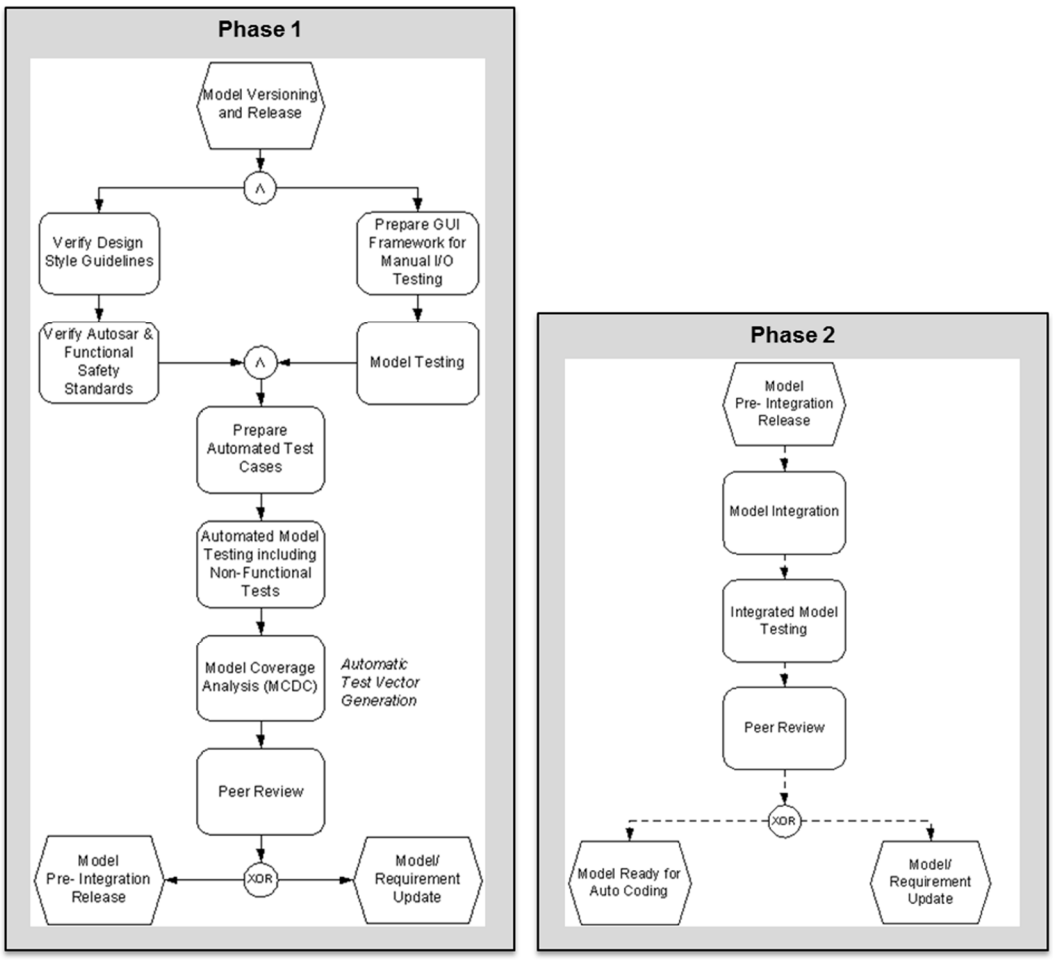


Figure 15. Level 1of the proposed MBPE process.

Modified Condition Decision Coverage (MCDC) is used to analyse the structural aspect of the model by executing auto generated test cases or vectors. MCDC coverage verifies that the model does not contain any uncovered or unapproachable sections. Coverage criteria are specified and documented in the requirements or in the modelling standards definition document. The coverage criteria depend on the Automotive Safety Integrity Level (ASIL) of the component being developed. A technical and process compliance peer review is conducted following the model verification and validation phase in Level 1.

The second phase of Level 1 involves the integration of all models that represent the feature or system under development. Integrated models are tested in a simulation environment (PC-based) in order to ensure that individual models function together as one system. If the vehicle feature under development is distributed across different components or ECUs, the functionality of other ECUs is modelled and integrated. Technical and process compliance peer review is conducted following the model integration and testing phase in Level 1. If structural or functional defects are identified during Level 1 then the process loops back to Level 0. The engineering activities and the associated deliverables from Level 1 are shown in Table 10.

Gateway	Deliverable	Input	Output
A	Compliance to modelling standards	• Functional model suitable for auto-coding • Design style guidelines • Naming conventions • Industry standards (AUTOSAR) • Functional safety standards (ISO26262)	• Report of compliance against modelling standards • Plan for addressing containment actions
A	Compliance to functional safety standard relating to model development	• Functional model suitable for auto-coding • Functional safety standards (ISO26262)	• Report of compliance against functional safety standard • Plan for addressing containment actions
A	Model validation	• Functional model suitable for auto-coding • Model design verification method	• Graphical user interface • Automated test cases • Report of test results
A	Robustness testing	• Functional model suitable for auto-coding • Model design verification method • Robustness testing engineering standard	• Report of test results
A	Model coverage and analysis (MCDC)	• Functional model suitable for auto-coding • Software requirements specification	• Report of MCDC coverage test results
A	Model integration	• Functional model suitable for auto-coding • Software requirements specification • System architecture diagram • Model design verification method	• Integrated model to support feature/function development and testing • Report of test results
A	Level 1 peer review	• All Level 1 deliverables	• Level 1 sign-off

Table 10. Level 1 high level deliverables.

Deliverables target specific PCS gateways to ensure development and initial verification and validation of the feature or system under test is conducted prior to supplier nomination or software release. Development engineers have clear set of deliverables driven by unambiguous input and output engineering activities. The engineering activities and deliverables at each Level of the process are tool independent and as a result do not rely on specific Commercial-off-the-Shelf (COTS) toolsets. The engineering activities and deliverables from all Levels of the proposed process and their link to specific PCS gateways are given in Submission 2.

4.1.4. Standardisation and applicability

The proposed process is standardised and tools independent. The outcome of the process standardisation was the creation of two engineering standards and two design rules as shown in Table 11. Details of the purpose and description of each engineering standard and design rule are given in Submission 2.

	Model-based Product Engineering Levels					
Engineering Standards and Design Rules	**0**	**1**	**2**	**3**	**4**	**5**
Engineering Standard A – Model-based Product Engineering	Yes	Yes	Yes	Yes	Yes	Yes
Engineering Standard B – Linking Models and Requirements	Yes	Yes	No	No	No	No
Design Rule A – Hardware-In-the-Loop Configuration	No	No	No	Yes	Yes	Yes
Design Rule B – Electronic Control Unit Integration for Hardware-In-the-Loop	No	No	No	Yes	Yes	Yes

Table 11. Engineering standards and design rules and their applicability on MBPE process.

Engineering standards and design rules were cascaded within JLR departments and supplier base where applicable. The process of cascade ensures consistency in the

development of embedded software and drives embedded software verification and validation at the early stages of the product development. Each engineering standard and design rule is coupled with a compliance matrix in order to ensure robust deployment and execution of the proposed process. A detailed deployment strategy including an evaluation of system complexity linked to the proposed MBPE process is described in Submission 2. An important factor to consider during deployment is the applicability of each Level of the process where there are shared activities between the Original Equipment Manufacturer (OEM) and suppliers. The applicability of each Level of the process against different embedded software development scenarios is shown in Table 12. These scenarios cover all possible routes which an OEM can follow when it comes to automotive embedded software development. As can be seen from Table 12 all scenarios have some applicable Levels apart from the case where a component does not have embedded software. More details can be found in Submission 2.

	Model-based Product Engineering Levels					
Embedded Software Development Scenarios	**0**	**1**	**2**	**3**	**4**	**5**
Embedded software developed in-house	OEM	OEM	OEM	OEM and/or Supplier	OEM	OEM
Functional models developed in-house and shared with suppliers for embedded software development	OEM	OEM	RCP at OEM SIL at Supplier	OEM and/or Supplier	OEM	OEM
Embedded software developed entirely by suppliers using model-based design	L0-A at OEM L0-B at Supplier	Supplier	Supplier	OEM and/or Supplier	OEM	OEM
Embedded software developed entirely by suppliers using non-model-based design	L0-A at OEM L0-B N/A	N/A	N/A	OEM and/or Supplier	OEM	OEM
Component does not have an embedded software	N/A	N/A	N/A	N/A	N/A	N/A

Table 12. MBPE process applicability for different embedded software development scenarios.

4.2. Generic design verification interface

Current research appears to validate the view that existing automotive test exchange standards do not address functional testing of embedded software as they are either not ready for implementation or mainly focus on vehicle diagnostics and exchange of simulation models. The development of standardised test exchange interfaces has the potential for better engineering resource utilisation in terms of efficiency and effectiveness. Efficiency and effectiveness of test resources such as test scripts, test engineers and test tools can drive detection of failure modes and/or software defects to the early stages of the product development.

In the automotive industry traditionally both manual and automated test cases are developed and used for testing automotive components, features or functions (Huang *et al.*, 2010), (Montazeri-Gh *et al.*, 2011), (Keranen and Räty, 2012), (Bansal *et al.*, 2013), (Altinger *et al.*, 2014). These test cases or sequences are usually developed for specific test tools and test targets on which they are being executed. Since these test sequences are designed and coded to match specific test tools and targets they cannot be further reused for other applications which require different set of test tools and test targets. In addition to that, very often engineers have to rewrite existing test cases in order to meet requirements of specific tool sets within their organisations. Therefore, current best practice results in the following problems:

- Long lead times to develop test scripts suitable for automated testing leading to late embedded software validation.
- Difficult to re-use existing test cases for different test tools. Lack of test scripts re-use leads in many challenges such as coping with late embedded software changes and test script updates.

- Lack of early testing before vehicle prototypes and Electronic Control Units (ECUs) are available. Currently test cases are written for real-time targets for testing ECUs with production embedded software rather than offline PC-based environments consisting of functional (or implementation) models.
- Lack of test scripts exchange and sharing between OEM and suppliers.

In order to overcome the above problems a new generic test interface called Design Verification Interface (DVI) has been proposed. A high level integration of MBPE process and DVI is shown in Figure 16. The generic DVI links with all the MBPE levels. The integration provides traceability and test script exchange and re-use across all MBPE levels.

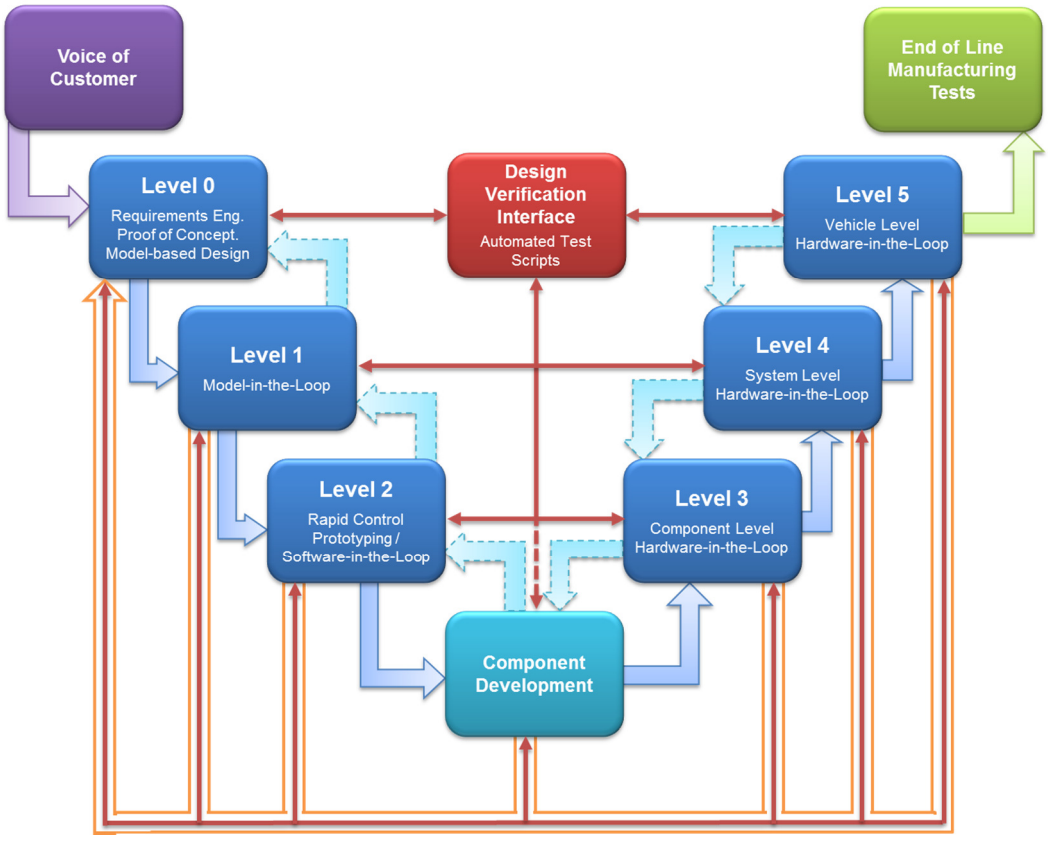


Figure 16. Integration of MBPE and DVI.

4.2.1. Requirements definition for the proposed generic test interface

A set of requirements, shown in Table 13, were defined for the development of the proposed generic DVI. These requirements are mainly target support for both offline and real-time test environments as well as support for signal profiles driving common and known automotive use cases. More details are provided in Submission 3.

Number	Requirement
1	Support of different signal profiles
2	Support of test case control flow (logical and conditional)
3	Support functional-based type testing
4	Support offline testing (MIL/SIL-based)
5	Support real-time testing (RCP/HIL-based)
6	Support in-vehicle networks diagnostics testing
7	Support all MBPE abstraction levels
8	Link to standardised DVM
9	Support auto-generation of test scripts
10	Plugin for MATLAB interface – PC-based testing
11	Plugin for Python (dSPACE) interface – Real-time-based testing
12	Proven with real automotive applications

Table 13. Requirements for the proposed generic DVI.

4.2.2. Design verification interface development

The proposed generic DVI has been inspired by the work on Functional Mock-up Interface (FMI) and Open Test eXchange (OTX) standards. Unified Modelling Language (UML) was used to define the DVI data engine. UML is a general purpose visual modelling language used to visualise, specify, construct and document software systems (Junior *et al.*, 2010), (Parreiras and Staab, 2010), (Kaur and Singh, 2011), (Weilkiens, 2011), (Petre, 2013), (Papajorgji and Pardalos, 2014). UML diagrams were used in order to enhance the readability of the DVI data engine. A short introduction to UML, eXtensible Markup Language (XML) and XML Schema Definition (XSD) is given

in Submission 3. A full mapping example shown in Figure 17 is used in order to explain how the DVI data engine is represented in UML, XSD and XML. In Figure 17 `<jaguarlandrover>` is the top level element. It inherits from `Company`. An unbounded list of element `<vehicle>` is allowed under `<jaguarlandrover>`.

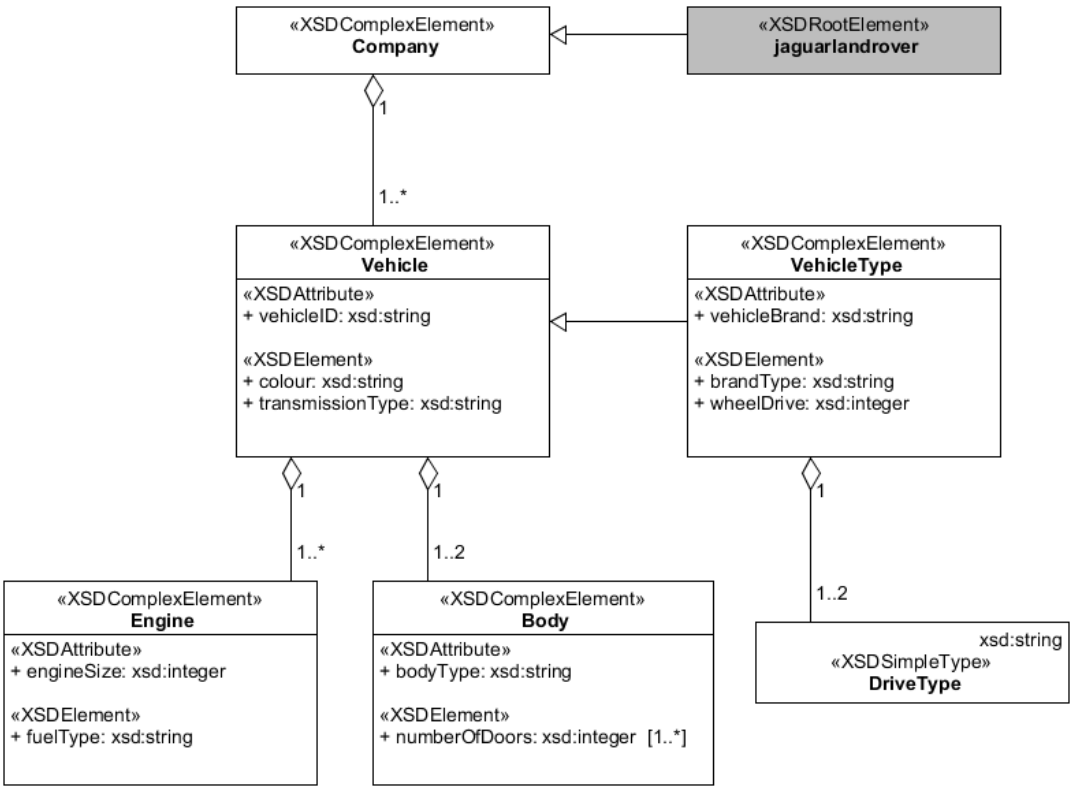


Figure 17. Full example of UML data model using XSD classes.

The `<vehicle>` class attributes are `vehicleId` of `xsd:string` and `vehicleBrand` of `xsd:string`. The `<vehicle>` class elements are `<colour>` and `<transmissionType>` of `xsd:string` type. `<Body>` is an `XSDComplexElement` and has aggregation relationship with `Vehicle` class with attribute `bodyType` as `xsd:string` and element `<numberOfDoors>` as `xsd:integer`. `<Engine>` is an `XSDComplexElement` and has aggregation relationship with `Vehicle` class with attribute `engineType` as `xsd:integer` and element `<fuelType>` as `xsd:string`.

`VehicleType` has inheritance relationship with `Vehicle` class with simple elements `<wheelDrive>` of `xsd:integer` and `<brandType>` of `xsd:integer` type and attribute `<vehicleBrand>` as `xsd:string. DriveType>` is a `XSDSimpleType` of `xsd:string.`

The full XSD model derived from the UML model shown in Figure 17 and the full XML document derived from the UML and XSD are shown in Appendix C.

A high level overview of the DVI engine data is shown in Figure 18.

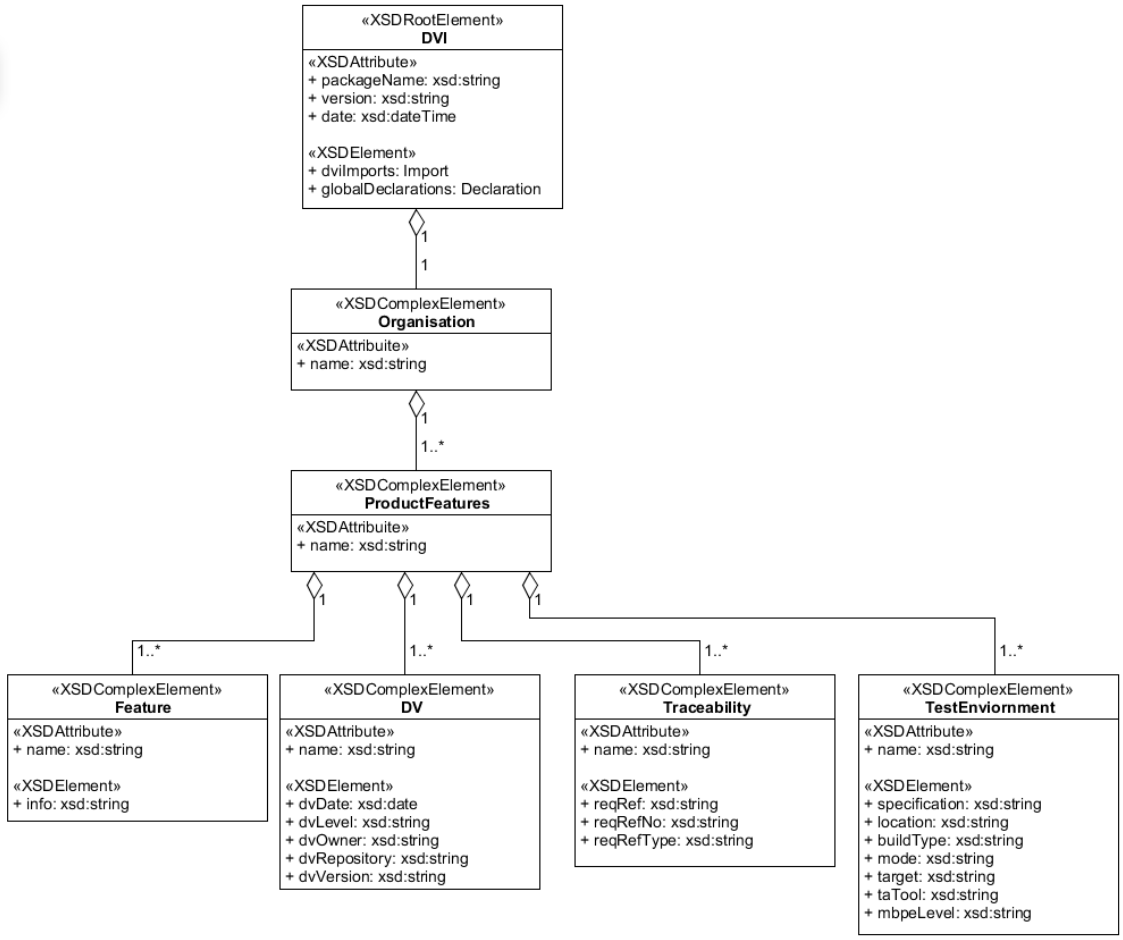


Figure 18. High level overview of the DVI engine data.

Figure 18 only contains a subset of types, attributes and relationships of the overall model. Nonetheless it reflects the main structure behind the DVI engine. As can be seen from Figure 18 seven classes define the backbone of the DVI engine. These are: `DVI`, `Organisation`, `ProductFeatures`, `Feature`, `DV`, `Traceability` and `TestEnvironment`.

Figure 19 shows the syntax of the `DVI` class.

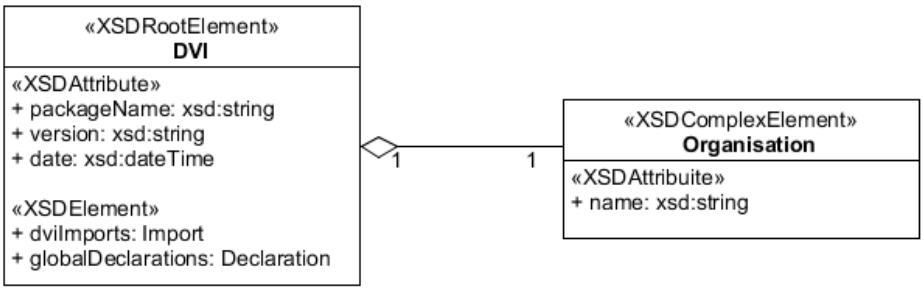


Figure 19. DVI class.

The `<dvi>` class has the following semantics.

- `packageName: xsd:string.` This attribute represents the package where the DVI file belongs to. The `packageName` type is an `xsd:string` simple type.

- `version: xsd:string.` This attribute contains the version of the DVI file (for supporting versioning systems). The generic DVI does not prescribe to any rules for versioning. It just defines a location where versioning information can be updated, if needed.

- `date: xsd:dateTime.` This attribute contains the DVI file creation timestamp. Standard format supported by the DVI is `yyyy-mm-dd T hh:mm: ss.`

- `<dviImports>: Import.` Contains a list of `<Import>` elements for importing other DVI files.

- `<globalDeclarations>: Declaration.` This element represents global declaration. This is the place where global constants and variables are defined.

A sample of the DVI file "`DVI.xml`" is shown as follows:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<dvi
    xmlns = "http://dvi.org/DVI"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    packageName = "DVI_Example"
    version = "1.0"
    date = "2015-07-15 T13:20:10" >
    <dviImports>
        <!--DVI elements-->
    </dviImports>
    <globalDeclarations>
        <!--DVI elements-->
    </globalDeclarations>
</dvi>
```

In Submission 3 each class of the DVI data engine is presented separately in terms of class description, syntax and semantics. In all cases an example is given to show the implementation of the class.

4.3. Standardised design verification method

In order to improve automotive embedded software quality and reduce development lead times the automotive industry must transition from manual to automated testing. As a result automated testing has become an attractive topic of interest for many researchers (Siegl *et al.*, 2011), (Börjesson and Feldt, 2012), (Alegroth *et al.*, 2013), (Iqbal *et al.*, 2015). The transition though from manual to automated testing involves the creation of test cases and test scripts that are suitable for automated execution in both offline and real-time test environments. In order to speed up verification and validation activities at the early stage of the product development it is imperative to develop new test specification methods for test case creation and automated generation of test scripts. Literature on specification approaches indicates that informal specification approaches can be ambiguous and open to interpretation. Furthermore, formal specification approaches are too complex to understand and implement. The use of semi-formal specification approaches has a potential if new methods can be developed that are suitable for the definition of vehicle test specifications. The capability of such a semi-formal specification approach can enable automated generation of test scripts directly from test specification documents.

4.3.1. Requirements definition

A set of requirements, shown in Table 14, were defined for the development of the proposed Standardised Design Verification Method (SDVM). The SDVM must support all key vehicle systems and DVI attributes. In addition to that the same method must support both manual and automated testing as well as all MBPE abstraction levels. Microsoft Excel was recommended as a test definition environment. The choice of Microsoft Excel is driven from the fact that it is widely available to most organisations and also known to most development engineers in the automotive industry.

Number	Requirement
1	Support the test definition of all key vehicle systems
2	Support all generic DVI attributes such as control flow, signal profiles, etc.
3	Support all MBPE abstraction levels
4	Support both offline testing (MIL/SIL-based) and real-time testing (RCP/HIL-based)
5	Support both manual and automated vehicle testing
6	Support test definitions in Microsoft Excel

Table 14. Requirements for the proposed standardised DVM.

4.3.2. Development of a new standardised design verification method

A typical vehicle electrical architecture contains up to 72 interconnected ECUs across different vehicle networks (such as CAN, MOST, LIN, Ethernet) and systems such as infotainment, human machine interface (HMI), climate, seats, body, safety, chassis, driveline and engine.

In order to identify the list of all attributes required for the development of the new Standardised Design Verification Method (SDVM) eleven existing vehicle test specifications were collected and analysed. The vehicle systems for these test specifications were: body electronics (interior lighting); powertrain electronics (accelerator pedal); comfort electronics (driver seat); climate electronics (auxiliary heater control); power supply electronics (load management); legacy infotainment electronics (AM/FM); new generation of infotainment electronics; hybrid control systems; chassis control systems; human machine interface (TV); transmission electronics. The above vehicle systems represent the whole vehicle electrical, electronics and software architecture for most premium vehicle manufacturers. A subset from each system test specification is shown Submission 4. The findings from the analysis has shown that all system test specifications were written in a completely

different format hence making it very difficult for test engineers to auto-generate and share test scripts. In addition to that, existing test specifications were written using informal specification approach which can be unclear and open to interpretation. Informal test specifications can lead to significant re-work and re-engineering of the embedded software as many tests can be implemented incorrectly hence causing late and expensive software changes. The analysis and assessment of existing test specifications led to the identification of key attributes for the development of the SDVM. The list of attributes was updated throughout the analysis in order to ensure that new attributes were captured and added in the analysis for the design of the new SDVM. The full list of attributes captured from the analysis is shown in Appendix D.

The data from the analysis from all eleven test specifications are confidential and therefore are only available in Submission 4. The data illustrate the complexity involved in the development of vehicle test specifications. Each test specification is driven from a different set of attributes. Over the years this level of complexity and the need to maintain legacy systems has driven engineers and suppliers in the creation of different types of types of test specification for each vehicle system, feature or function. The findings from the analysis also suggest that most vehicle systems are validated using vehicle prototypes or in a real-time test environment (HIL testing). This is mainly driven from the fact that the capability and the processes in the industry to allow validation and execution of test cases offline, are not currently in place. Enabling offline test case execution for all vehicle systems, features and functions will move software validation and verification to the early stages of the vehicle development process. This has the potential to allow the detection of failure modes and/or software defects at earlier phases of the automotive embedded software development process.

The new SDVM is an Excel based template where feature owners and test engineers can enter vehicle system or vehicle feature test cases using semi-formal specification representation. The front sheet of the SDVM template is shown in Figure 20. As can be seen from Figure 20 the SDVM template contains eight sections.

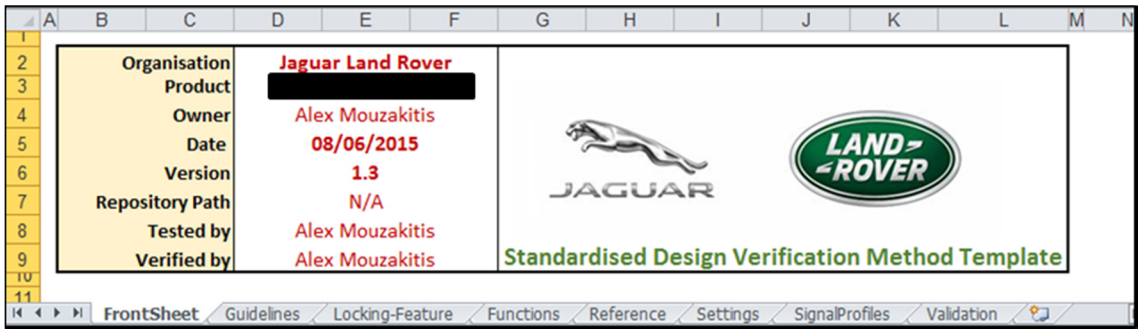


Figure 20. *FrontSheet* section of the SDVM template.

The front sheet of the DVM template contains high level test specification details mainly related to the ownership of the vehicle system or feature. The guidelines section contains the rules of how a user should populate the SDVM template. The x-Feature (Locking-Feature in this example) section is the most important part of the SDVM template. It contains the full list of test cases related to a particular vehicle feature. A sub-set and an example of the x-Feature section containing test cases related to locking vehicle feature is shown in Figure 21. The user can define test cases using a semi-formal specification approach. The section titled *"purpose of the test"* defines the description of the test case using informal specification approach. Here full description of the test case using free text is allowed. The *"purpose of the test"* section can be used for manual as well as for automated testing. The definition of input and output signals including signal profiles is defined using semi-formal and formal approach. In the section titled *"input signal name"* the user can select a signal from a predefined signal list. The signal name uses a semi-formal specification approach as free text is not allowed. In the section titled *"input signal value"* the user can select a signal value

from a predefined values derived from mathematical formulae. Figure 22 shows an example where the signal type has a pulse profile. The section titled *"condition/branches"* defines the conditioning and branching of the specified test cases. A full description of the supported condition and branches and an example is given in Appendix E.

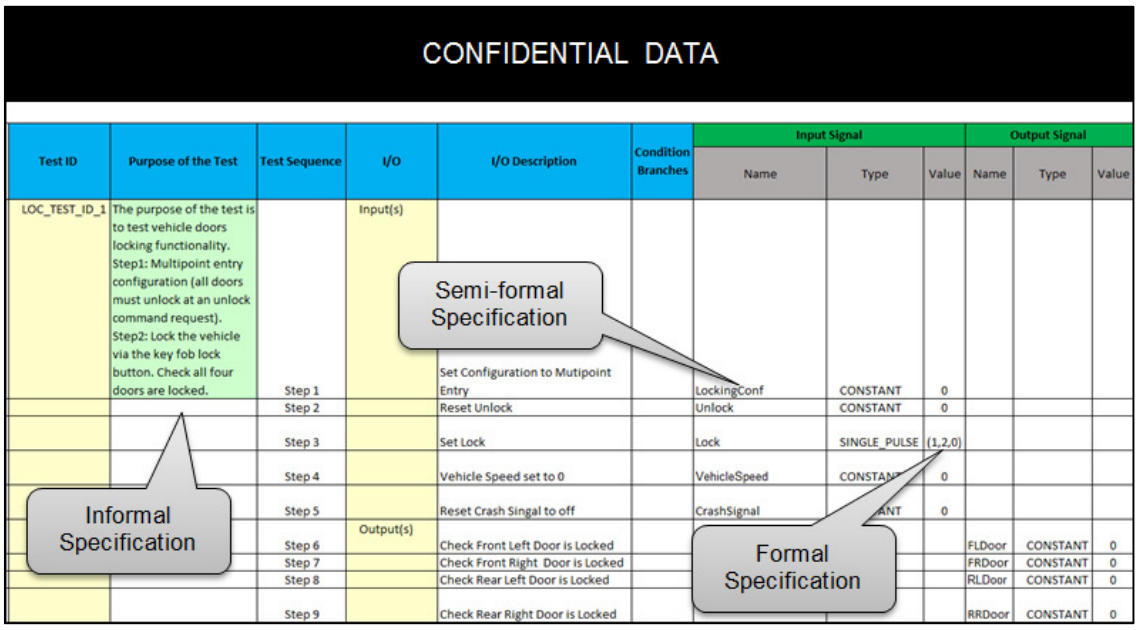


Figure 21. Sample of the *Locking-Feature* section of the SDVM template.

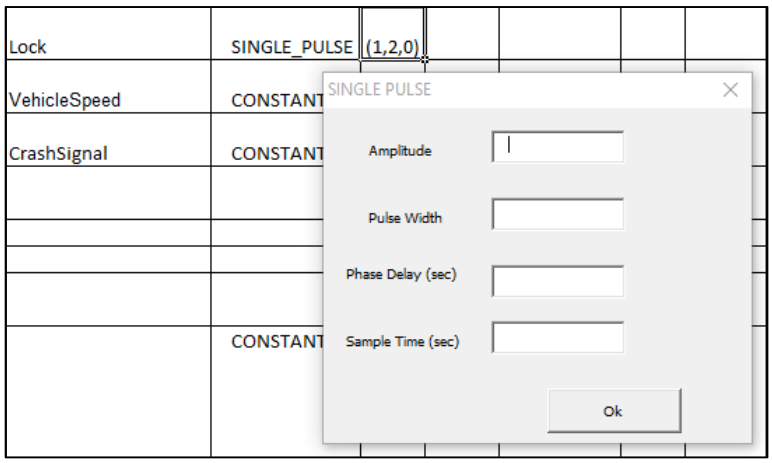


Figure 22. Definition of a signal profile using formal specification.

The remaining sections of the SDVM template are:

- Functions, is used to define test cases that can be called from the main feature section;
- Reference, is used to provide a full mapping between the input and output test signals and the I/O of the system under test;
- Settings, contains configurations and general settings for the test target such as MATLAB and dSPACE;
- SignalProfiles, is used to pre-define a set of signal profiles accessible within the SDVM template for the test case definition;
- Validation, is used to determine whether the SDVM has been populated correctly.

More details and full description of the proposed SDVM template are given in Submission 4.

In summary the SDVM template allows test cases from all vehicle systems to be written in both informal and a semi-formal standardised forms. The SDVM template enables test cases to be presented in a text free form as well as machine readable data. The semi-formal standardisation has the potential to enable an automatic extraction of test scripts suitable for execution in both offline and real-time test environments.

4.4. Platform independent test system

The development of a holistic end-to-end solution to support automation across all levels of system abstraction with reduced engineering effort and time is presented in this sub-section. A proposed solution called Platform Independent Test System (PITS) utilises and integrates both the generic DVI and SDVM. The intent benefits of the proposed solution are to:

- Provide an end to end solution that can be used throughout the product development cycle from the requirements stage to the automated test case execution (offline and real-time environments are supported).
- Enable auto-generation of test target (or platform) independent test scripts directly from design verification documents.
- Reduce or eliminate the need of Commercial-off-the-Shelf (COTS) test automation tools, and offer JLR and its suppliers an environment where test scripts can be auto-generated and executed at all levels of abstraction of the MBPE process.
- Reduce the engineering effort, cost and time required for the definition of test cases, the generation of test scripts and the execution of test scripts in both offline and real-time test environments.

Reduction of engineering effort and time together with the execution of automated test scripts at the early stages of the MBPE process (Levels 1, 2 and 3) has the potential to shift failure modes and/or software defects detection early in the automotive product development.

Figure 23 shows an overlay of the PITS applicability against the MBPE process. As can been seen from Figure 23 all MBPE process levels are covered by the introduction of PITS.

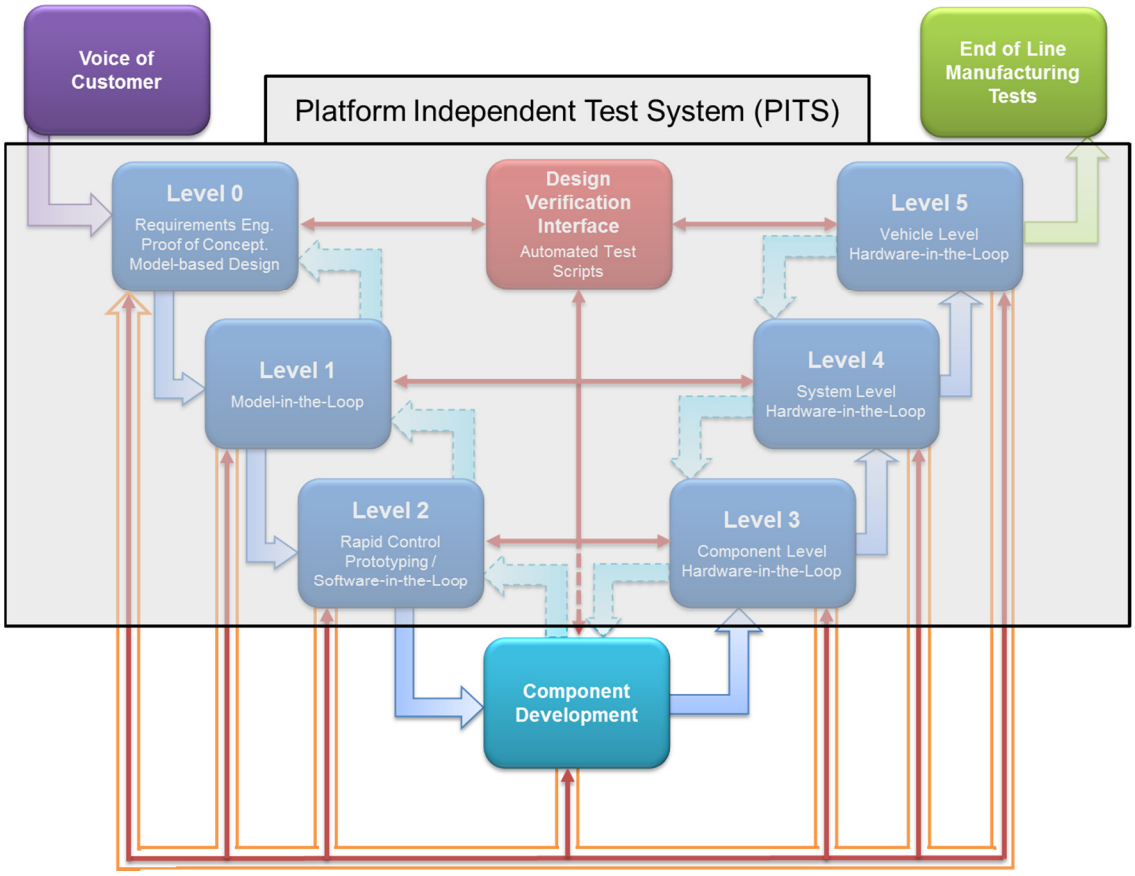


Figure 23. Mapping of Platform Independent Test System (PITS) against the Model-based Product Engineering (MBPE) process.

An abstract system representation of the proposed Platform Independent Test System (PITS) depicting the various inputs and outputs is shown in Figure 24. The primary function of the proposed PIT system is to auto-generate platform independent test scripts directly from Standardised Design Verification Method (SDVM) test specifications. The user, in this case a test engineer, selects an SDVM test specification for a given vehicle system or feature and its associated functions. Once an SDVM has been selected the user then selects the offline or real-time environment

(test platform or test target) for the auto-generated test scripts to be executed. A typical offline test platform is MATLAB and a typical real-time platform is dSPACE.

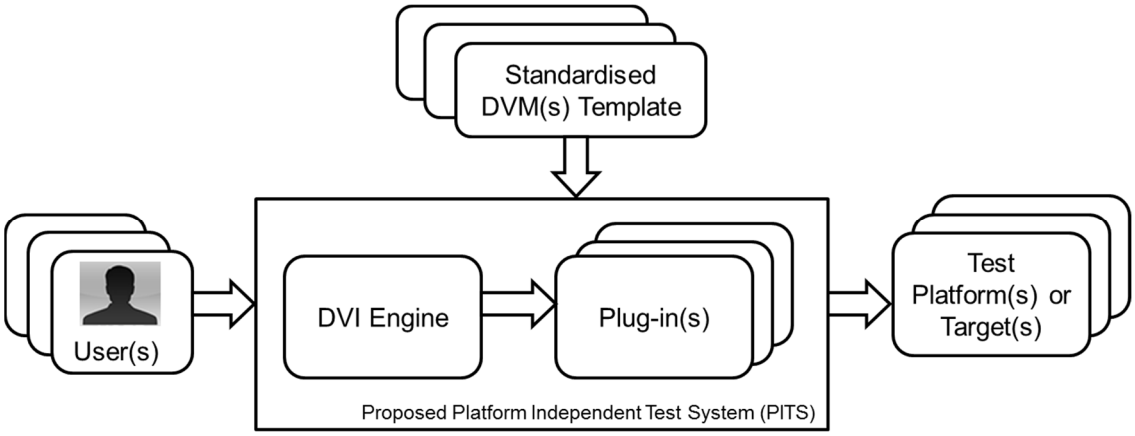


Figure 24. Abstract representation of the proposed Platform Independent Test System (PITS).

The execution of automated scripts relies on the presence of the generic Design Verification Interface (DVI) together with a number of plug-ins. Plug-ins are used in order to link the auto-generated test scripts with the associated test platforms or test targets. More details on plug-ins and their benefits are given in Submission 4.

4.4.1. Methodology and design approach

The high level methodology and design approach of the proposed PIT system is depicted in Figure 25. The detailed methodology and design approach of PITS are given in Submission 4.

The first step involves the identification of PITS stakeholder (or customer) requirements. Once stakeholder requirements have been identified the next step is to identify the key use cases that cover all stakeholder requirements. Use case analysis is

an important step in requirements definition as it ensures that all key actors (human, machine or process inputs) are identified for the proposed system.

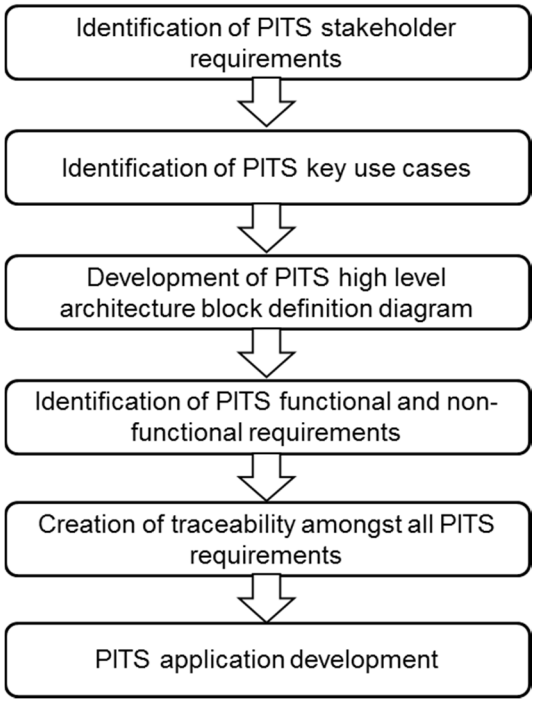


Figure 25. High level methodology and design approach of PITS.

The development of a high level architecture Block Definition Diagram (BDD) follows the identification of stakeholder requirements and use cases. The primary purpose of the BDD is to communicate structural information of the proposed PIT system.

Once a high level architecture has been defined the next step is to identify both functional and non-functional requirements. A traceability matrix was then developed in order to link together stakeholder requirements, use cases, high level architecture block definition diagram and functional/non-functional requirements of the proposed PIT system. The final step of the design approach was the development and prototype of the PITS Windows-based application.

4.4.2. Stakeholder requirements

Capturing stakeholder (or customer) requirements at the start of the project can help to deliver a final product that has a higher potential to meet all customer expectations. In the literature and in particular in the area of Requirements Engineering (RE), the process of capturing and analysing stakeholder requirements is well understood (Hull *et al.*, 2011), (Chemuturi, 2013). Stakeholder requirements consist of requirements driven by the user, the sponsor and the operator. In this research, the user requirements are realised by the role of a test engineer. The sponsor requirements are realised by the role of senior management. The operator requirements are realised by the other associates to the project or organisation. In this case the operator requirements are driven from vehicle system owners and feature owners. Table 15 shows a sample of the stakeholder requirements for PITS development. Full stakeholder requirements are given in Submission 4.

Number	Description	Stakeholder
STK_SRS_1	The system shall provide an interface for the creation and automated execution of test cases suitable for all Model Based Product Engineering (MBPE) levels (Level 0 to Level 5).	Sponsor
STK_SRS_2	The system shall automatically generate platform independent test scripts directly from standardised Design Verification Method (SDVM) test specifications.	Sponsor
STK_SRS_3	The system shall provide a generic Design Verification Interface (DVI) suitable for test exchange and integration with different test automation environments (support for offline and real-time test targets).	User
STK_SRS_4	The system shall have minimum support configuration for dSPACE (real-time) and MATLAB (offline) test environments and plug-ins.	User
STK_SRS_5	The system shall support configuration for monitoring and logging of in-vehicle network signals.	User
STK_SRS_6	The system shall automatically generate platform dependent code to support the execution of automated test script(s) on selected test targets.	User
STK_SRS_7	The system shall support a generic signal profiles library for the creation of test case(s).	Operator

Table 15. Sample of stakeholder requirements for PITS.

Stakeholder requirements have been captured through interviews with test engineers, system owners and features owners. These interviews targeted all eleven vehicle systems used for the development of the SDVM template. Stakeholder requirements were numbered for traceability purposes with a unique identification number. Stakeholder requirements were numbered as follows:

STK_SRS_ID

Defining, STK as stakeholder, SRS as software requirements specification and ID as a requirement unique identification number.

4.4.3. Development of use cases

Once stakeholder requirements were captured the next logical step in requirements engineering process was to define the use cases of the system or product under development (Dorfman and Thayer (eds.), 2000). In this research SysML use case diagrams were used to depict the main use cases for the development of PITS. In requirements engineering use case diagrams represent the highest level of abstraction available to the designer, developer or customer. In addition to that use case diagrams are very powerful for communicating to all stakeholders the main interactions and operations of the system under development.

Figure 26 shows a high level use case for the PIT system. The content of the use case is defined in the system boundary as *"PIT System Use Case"*. The relevant actors are shown as *Test Engineer*, *Feature Owner* and *Test Target*. *Test Engineer* and *Feature Owner* actors are roles within JLR product development whereas the *Test Target* actor is a test environment influencing the development and execution of PITS.

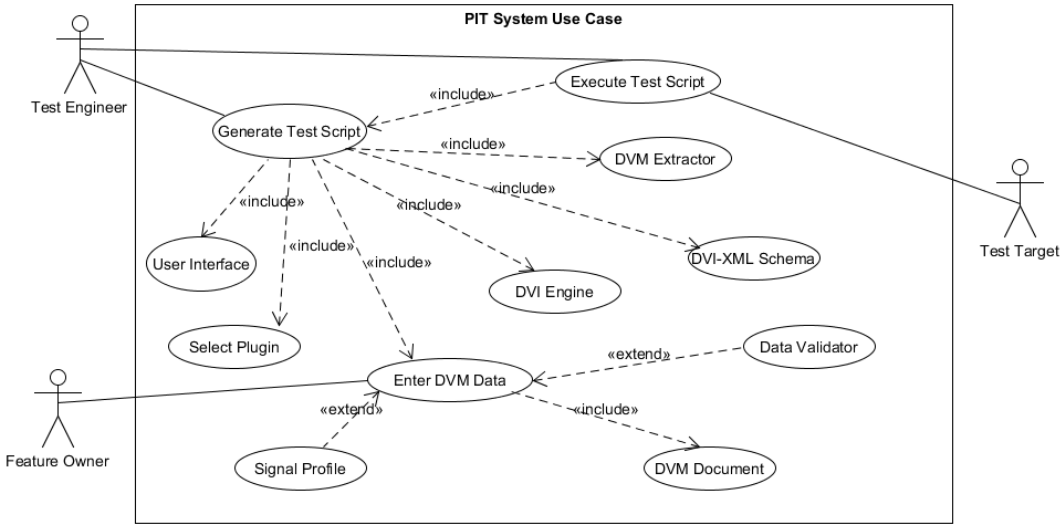


Figure 26. High level use case for PITS.

The *Test Engineer* actor is connected to *"Generate Test Script"* and *"Execute Test Script"* use cases. The *Test Target* actor is connected to *"Execute Test Script"* use case and the *Feature Owner* actor is connected to *"Enter DVM Data"* use case. *"<<include>>"* and *"<<extend>>"* type of relationships were used between use cases. As an example, *"Data Validator"* use case extends *"Enter DVM Data"* use case. The *"<<Extend>>"* relationship causes a change in the main use case. Here, the *"Data Validator"* software will cause a change to the *"Enter DVM Data"* process and hence affect the role of *Feature Owner* and its interaction with the PIT system.

A full description of each use diagram is given by a table which defines the name of the use case, the use case ID, the primary actors, the description of the use case, the pre-condition and post-condition of the use case and the traceability link between a given use case and stakeholder requirements.

A full description of all uses cases for the PITS development is given in Submission 4.

4.4.4. System architecture

The definition of the high level system architecture Block Definition Diagram (BDD) helps to communicate structural information about the proposed PIT system (Holt *et al.*, 2011), (Chemuturi, 2013). In addition to that BDD communicates relationships that exist within the proposed system. Relationships link key conceptual components of the system under development.

The system architecture as a BDD representation is depicted in Figure 27. The diagram in Figure 27 makes use of aggregation to show the structure of PITS. The aggregation is represented by the use of a diamond at the end of the relationship. The diagram in Figure 27 is read as follows: The *'PIT System'* is made up of *'Existing DVMs'*, *'Common DVM Template'*, *'DVM Converter'*, *'DVI Data'* and *'Test Script Execution'*. Aggregations can take role names and multiplicities (Wiegers and Beatty, 2013a). Common multiplicities used in this research are as follows:

0..1 *Indicates an optional value*

1 *Indicates exactly 1*

*0..** *Indicates any number, including 0*

** Same as 0..**

*1..** *Indicates 1 or more*

As an example, in the case of diagram Figure 27 the use of multiplicities in the aggregations was used as follows. Any number of *'Existing DVMs'* (DVMs from different vehicle systems) can be entered into one *'Common DVM Template'*. Additionally, *'PIT System'* can have one or more *'Test Script Execution'*. Full

description of all key components of the system architecture diagram depicted in Figure 27 is provided in Submission 4.

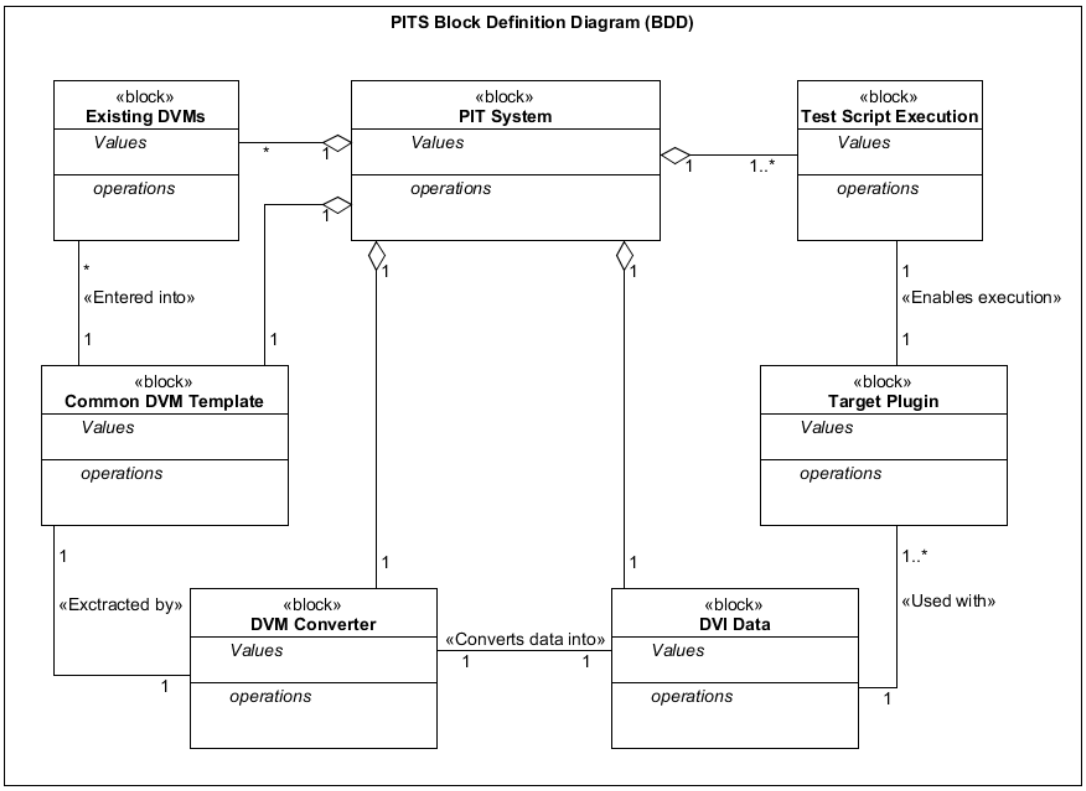


Figure 27. System architecture of the proposed PIT system.

4.4.5. Functional requirements

The development of Functional Requirements (FRs) along with the development of Non-Functional Requirements (NFRs) is an important and significant step in software development.

In this research requirements were classified using the FURPS+ (Functionality, Usability, Reliability, Performance, and Supportability plus Design, Implementation, Interface and Physical) model (Grady, 1992), (Kruchten, 2004). The FURPS+ model is widely used in industry and is proven to provide a good structure categorising requirements (Umar and Khan, 2012). The definition of the FURPS+ model in given in

Appendix F, in addition, more details are given in Submission 4. Functional requirements specify something that the system under development should or shall do, (Laplante, 2013b), (Wiegers and Beatty, 2013b). Typically, functional requirements are linked to behaviour of a system or function. An example of a functional requirement is *"The PIT system shall generate a generic XML representation of the SDVM data as an output".* Functional requirements tend to have verbs that reflect direct action in their definition. Functional requirements normally are solution-independent in order to be implementable by any engineering and technology software or hardware development.

Table 16 shows a sample of PITS functional requirements. Each functional requirement has a unique identification number and traceability to system architecture BDD and stakeholder requirements (for simplicity only link to BDD is shown). The full list of functional requirements of the PIT system is shown in Submission 4.

Number	Functional Requirement Description	BDD Number
FR_SRS_01	The PIT system shall generate a generic XML representation of the SDVM data as an output.	1, 4, 5
FR_SRS_02	The DVI generic XML data shall be the intermediate representation of the test data.	5
FR_SRS_03	The DVI generic XML data shall be the input for the auto-generated platform dependent test script.	5, 6
FR_SRS_04	The DVI generic XML schema shall contain product feature, DV, traceability and test environment details.	5
FR_SRS_05	The DVI generic XML data product feature shall contain function, component, system and product details attribute data which are extracted from the SDVMs.	5
FR_SRS_06	The DVI generic XML schema shall contain date, name, version, repository and owner details.	5
FR_SRS_07	The DVI generic XML traceability shall contain requirement reference, reference number and requirement type.	3, 5
FR_SRS_08	The DVI generic XML test environment details shall contain test scenarios, location, build type and test level.	3, 5

Table 16. Sample of PITS functional requirements.

4.4.6. Non-functional requirements

Non-functional requirements (NFRs) describe how the system works or the way that the functional requirements (FRs) are realised (Chung *et al.*, 2012). NFRs are often seen as quality attributes of a system under development (González-Huerta *et al.*, 2012). An example of a non-functional requirement is *"The PIT system shall be designed to run on Windows XP, 7 and 8"*. The eight categories of the FURPS+ model were used to define the type of the NFRs of the PIT system.

Table 17 shows a sample of PITS non-functional requirements. Each non-functional requirement has a unique identification number and traceability to system architecture BDD and stakeholder requirements (for simplicity link to BDD is not shown). In addition to that each NFR has link to FURPS+ model. FURPS+ ensures good coverage of NFRs and hence driving design completeness. The full list of non-functional requirements of the PIT system is shown in Submission 4.

Number	Non-Functional Requirement Description	Stk. Req.	Type
NFR_SRS_01	The PIT system shall generate an XML representation of the DVM data as an output.	STK_SRS_2	Implementation
NFR_SRS_02	The DVI XML data shall be the intermediate representation of the test data.	STK_SRS_2	Implementation
NFR_SRS_03	The DVI generic XML shall be developed to support STJLR- 18-063 engineering standard.	STK_SRS_1 STK_SRS_2	Supportability
NFR_SRS_04	The DVI generic XSD shall follow W3C standards for XML schema.	STK_SRS_1	Supportability
NFR_SRS_05	A MATLAB plugin shall support MATLAB version 2013b and 2014a.	STK_SRS_1 STK_SRS_3	Implementation
NFR_SRS_06	A dSPACE plugin shall support ControlDesk and ControlDesk Next Generation version 2013 in order to comply with JLR's current versions of toolset.	STK_SRS_1 STK_SRS_4	Implementation
NFR_SRS_07	The development framework shall be .Net Framework 3.5.	STK_SRS_1 STK_SRS_4	Implementation
NFR_SRS_08	The PIT system application shall extract the generic XML and automated test script without noticeable delay.	STK_SRS_4 STK_SRS_11	Performance
NFR_SRS_19	The PIT system shall be designed to run on Windows XP, 7 and 8.	STK_SRS_6	Design

Table 17. Sample of PITS non-functional requirements.

4.4.7. Traceability

Traceability amongst all type of requirements drives design completeness and ensures that stakeholder requirements are realised in a controlled and well-engineered system development environment (Wiegers and Beatty, 2013b).

Table 18 shows a sample of full traceability amongst all type of requirements, BDD and use cases of the PIT system. Each traceability line item has a unique identification number and traceability to a unique stakeholder requirement, use case(s), part(s) of the system architecture BDD, functional requirement(s) and non-functional requirements(s). The full list of traceability is given in Submission 4.

Number	Stakeholder Requirement ID	Use Case ID (UC_PITS_####)	BDD Number	Functional and Non-Functional Requirements ID (FR_SRS_### and NFR_SRS_@@@)
1	STK_SRS_1	001	1	FR_SRS_01, #18, #27, #56, #60, #61, #62, #63, #64, #65, #66, #67, #68, #69, #70, #71, #72, #73, NFR_SRS_03 to @07
2	STK_SRS_2	001, 004, 005	1, 4, 5, 6, 7	FR_SRS_01 to #18, #27, #56, #60, #61, #62, #63, #64, #65, #66, #67, #68, #69, #70, #71, #72, #73, #03, #14, #31 to #35, #44 to #63, #70, #71, NFR_SRS_01, @02, @03, @17, @18
3	STK_SRS_3	001, 003, 004, 005	1, 4, 5, 6, 7	FR_SRS_01 to #18, #27, #56, #60, #61, #62, #63, #64, #65, #66, #67, #68, #69, #70, #71, #72, #73, #03, #14, #31 to #35, #44 to #63, #70, #71, NFR_SRS_11, @16
4	STK_SRS_4	002, 004, 005	2, 3, 5, 6, 7	FR_SRS_01 to #18, #27, #56, #60, #61, #62, #63, #64, #65, #66, #67, #68, #69, #70, #71, #72, #73, #30 to #35,#44 to #63, #70, #71, NFR_SRS_06 to @09, @12, @13, @17, @18
15	STK_SRS_15	001, 002, 005	2, 3	FR_SRS_30, #31, #16, #18 to #32, #35 to #41, #58, #66

Table 18. Sample design traceability matrix of the PIT system.

4.4.8. Windows application

The next step in the research was the development of the front-end of the PITS Windows application shown in Figure 28. This screenshot is what the user sees once the PITS application has successfully loaded. The user selects a Standardised Design Varication Method (SDVM) template and the test target (offline or real-time) for the execution of the automated test scripts.

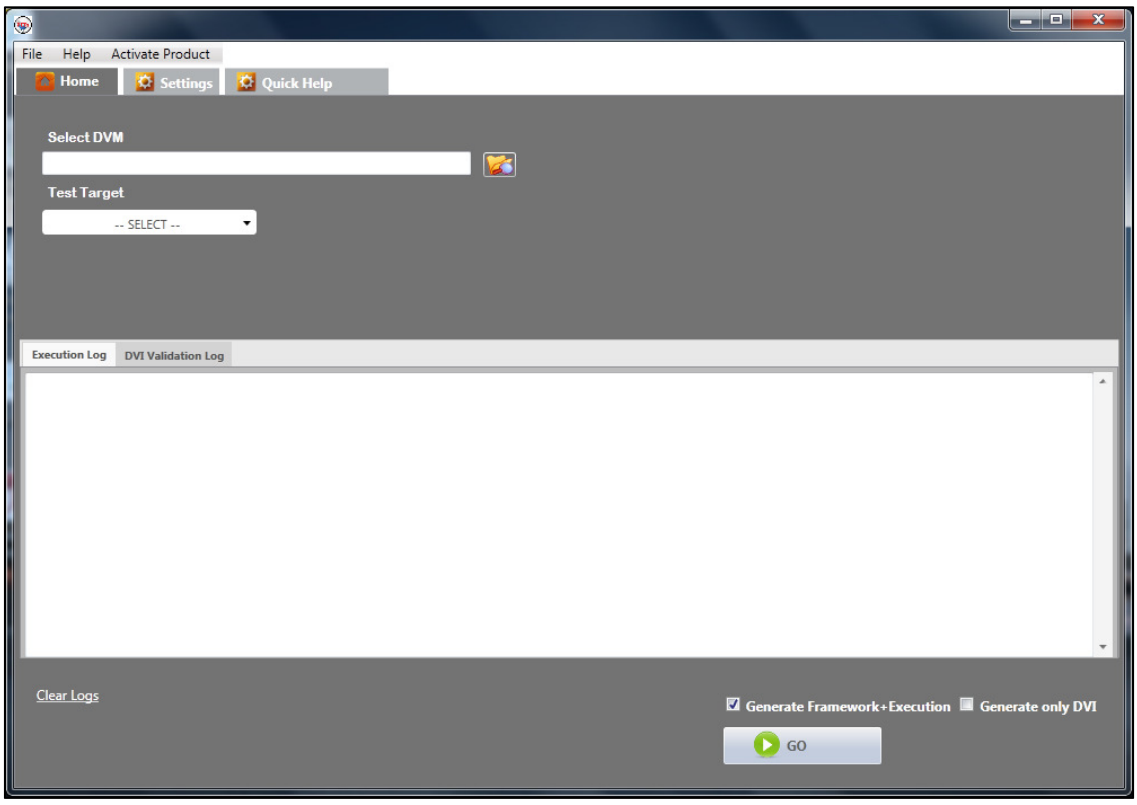


Figure 28. Main window of the PIT system application.

Two test targets are available for selection, the first is for MATLAB (offline automated testing) and the second is for dSPACE (real-time automated testing). The Settings tab in the main window allows the user to configure the settings of the test target and also to select specific vehicle features from the SDVM template if required. Full description of the PITS windows application is given in Submission 4.

5. Evaluation of research

This section is focused on an evaluation of the proposed solutions for automotive embedded software development. The evaluation of the research was based on a selection of evaluation criteria mainly driven from the initial design requirements of each proposed solution and a number of measures to assess effectiveness. The evaluation includes deployment of solutions within Jaguar Land Rover (JLR) where appropriate, and a number of vehicle case studies are used. Where appropriate comparison studies were conducted against other Commercial-off-the-Shelf (COTS) tools. The outcome of the evaluation is discussed and concluding remarks are drawn about the effectiveness of the proposed solutions against the evaluation criteria and research objectives.

5.1. Evaluation criteria

A set of key criteria were defined for the evaluation of the proposed solutions as shown in Table 19. The aim of each criterion was to ensure that either design requirements or measures driving business benefits for each solution are satisfied. The rationale behind the selection of the design requirements for each solution was defined in Section 4 and in the portfolio submissions. Business benefits and measures are drawn from the original motivation and objectives for this research. In Table 19 each criterion is associated with a proposed solution in order to channel the focus of the evaluation. However, in some cases evaluation criteria have a more holistic impact due to the integrated nature of the proposed solutions.

More details and a full description of the evaluation criteria defined in Table 19 can be found in Submissions 2, 3, 4 and 6.

Number	Criterion	Driven By	Solution
1	Satisfy MBPE process requirements listed in Table 9	Design Requirement	MBPE
2	Satisfy DVI requirements listed in Table 13		DVI
3	Satisfy SDVM requirements listed in Table 14		SDVM
4	Test target (or platform) independent		PITS
5	Satisfy PITS requirements (stakeholder, functional and non-functional)		PITS
6	Support of different signal profiles as defined in the DVI and SDVM		PITS
7	Support of offline and real-time automated testing		PITS
8	Capability to support all MBPE levels		PITS
9	Sustainable solution		PITS
10	Early detection of failure modes and/or software defects	Measure	MBPE
11	Efficiency in terms of engineering effort and time		PITS
12	Low cost		PITS
13	Low maintenance		PITS
14	Overall comparison against COTS tools		PITS

Table 19. Evaluation criteria driven by design requirements and measures.

5.2. Evaluation results

Evaluation results for each evaluation criterion shown in Table 19 are presented in the following sub-sections.

5.2.1. Criterion #1 – Satisfy MBPE process requirements

Table 31 in Appendix G.1 shows a summary of the evaluation results including which requirements have been satisfied and how. All MBPE requirements have been satisfied mainly through the creation of new engineering standards and design rules. In addition to that MBPE requirements have been satisfied through the creation of the generic Design Verification Interface (DVI), Standardised Design Verification Method (SDVM) and integration of MBPE with the JLR Product Creation System (PCS). The MBPE process is standardised and COTS tool independent. The process fully defines the engineering activities for both offline and real-time model-based automated testing as

defined by the engineering standards and design rules. Automated testing is tool independent and is driven, but not coupled, by the generic DVI and SDVM. The use of SDVM and generic DVI allows JLR engineers from different departments and the supplier base to write, auto-generate and share test requirements for automated or manual testing across all vehicle abstraction levels. Support of product complexity is covered at the early levels of the MBPE process together with a pragmatic deployment strategy as presented in Submission 2. Deployment strategy is driven by product complexity assessment, resource needs and tool support. The process was fully integrated with JLR's software Statement of Work (SoW) and JLR's Software Engineering Journal (SEJ) documents. Both engineering documents were fed directly to JLR supplier base for software sourcing, creation and standards compliance.

5.2.2. Criterion #2 – Satisfy DVI requirements

Table 33 in Appendix G.3 shows a summary of the evaluation results including which requirements have been satisfied and how. All DVI requirements have been satisfied and proven mainly through the integration of PITS, DVI and SDVM. In addition to that requirements have been satisfied through the auto-generation of test scripts for two vehicle features (locking and wiper control), both executed in offline and real-time environments. The evaluation of criteria 11 and 12 cover the details for both vehicle features. Furthermore, some DVI requirements were satisfied through the evaluation of the SDVM. The DVI solution was capable of handling different vehicle DVMs from different MBPE levels written in a SDVM. Comparison amongst the DVI and other automotive related standards is shown in Table 34 in Appendix G.3. The DVI cannot support ECU diagnostics testing and simulation models integration including co-simulation. These requirements are fully satisfied by the use of OTX, ODX and FMI standards. In addition to that the DVI interface is not an international standard in

comparison to OTX and ODX. More details on DVI comparison with other standards are given in Submission 3.

5.2.3. Criterion #3 – Satisfy SDVM requirements

A summary of the evaluation results for the proposed SDVM is given in Table 20. The evaluation of the SDVM was based on the test case definition driven by 12 key vehicle systems/features (see Table 35 in Appendix G.4). These systems/features covered the majority of the vehicle embedded software implementation in terms of complexity and definition of test cases. Test cases from all chosen vehicle systems were successfully implemented in the proposed SDVM and furthermore auto-generated/converted into a DVI XML machine readable form. In addition to that the chosen vehicle systems/features covered all MBPE process levels. More details of the evaluation can be found in Submission 6.

Number	Requirement	Satisfied	How
1	Support the test definition of all key vehicle systems	Y	12 vehicle systems/features were chosen as shown in Table 35 in Appendix G.4. All vehicle systems/features were successfully implemented in the proposed SDVM.
2	Support all generic DVI attributes such as control flow, signal profiles, etc.	Y	12 vehicle systems/features were successfully implemented in the proposed SDVM and auto-generated/converted into a DVI XML form.
3	Support all MBPE abstraction levels	Y	Chosen vehicle systems/features covered all MBPE process levels (see Table 35 in Appendix G.4).
4	Support both offline testing (MIL/SIL-based) and real-time testing (RCP/HIL-based)	Y	Satisfied in the evaluation criteria 11 and 12.
5	Support both manual and automated vehicle testing	Y	Satisfied in the evaluation criteria 11 and 12 and requirement 1 in this Table.
6	Support test definitions in Microsoft Excel	Y	See Appendix A in Submission 6.

Table 20. Summary of SDVM requirements satisfaction.

5.2.4. Criterion #4 – Test target (or platform) independent

A key benefit of the proposed PITS solution is being test target or test platform independent. Figure 29 depicts a process flow that shows how a test definition in a SDVM can be used on two different test targets through PITS.

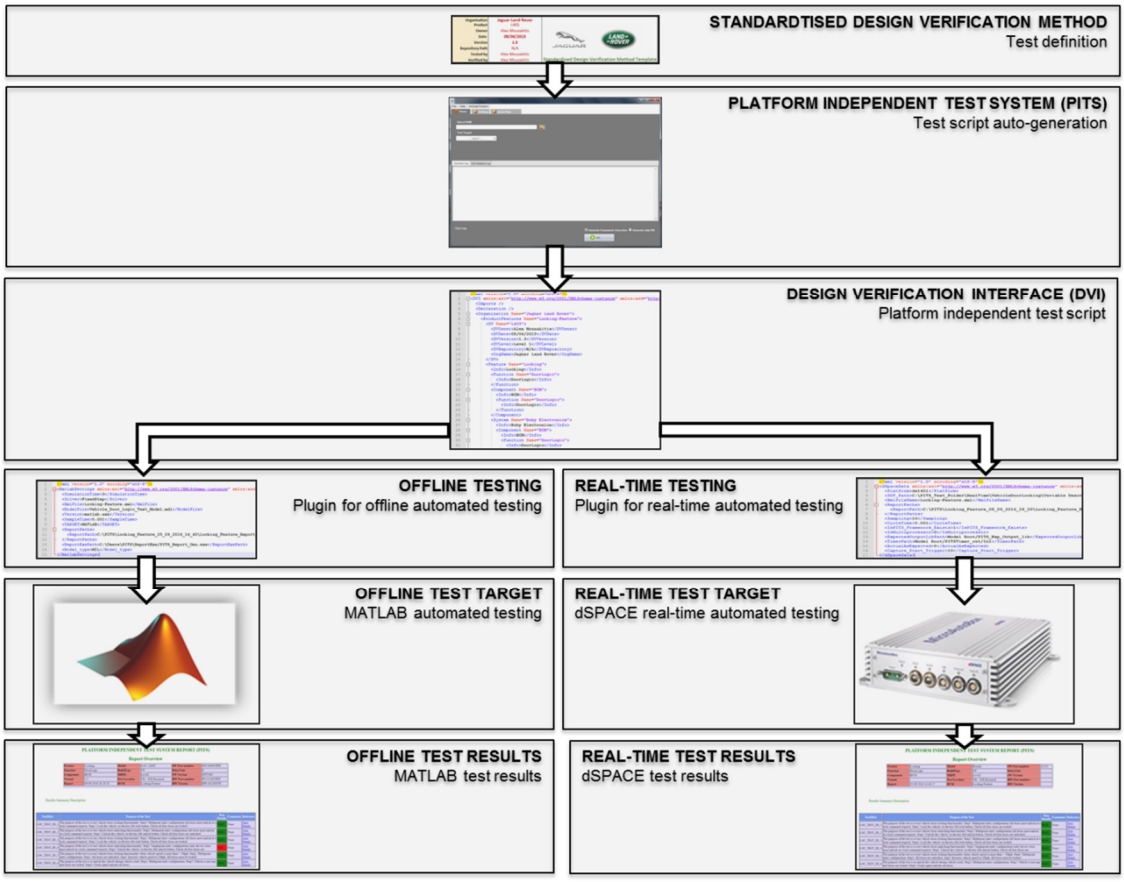


Figure 29. Overview of platform independent test system solution.

The PIT system reads a test definition written in SDVM and automatically generates platform independent test scripts through the DVI. At this stage, test scripts are platform independent and as a result can be used on any test target or test platform assuming a plugin is available. The DVI file at this stage can be archived for future use or can be shared with other stakeholders such as suppliers and other internal JLR departments. A plugin then is used in order to link the auto-generated test scripts to a

specific test target or test platform. After the execution of the automated test scripts is complete the PIT system generates a test results report as an HTML file. The interface to the test results report is identical for both test targets. Test results can be shared as a single file and can be viewed using a standard web browser (such as Explorer or Chrome). More details can be found in Submission 6.

5.2.5. Criterion #5 – Satisfy PITS requirements

Stakeholder, functional and non-functional requirements for PITS were satisfied apart from those listed in Table 21. A sample list of how PITS requirements were satisfied is shown in Table 36 in Appendix G.5. Stakeholder (15 in total) and functional (73 in total) requirements were mainly satisfied through the development of SDVM and DVI. Non-functional (19 in total) requirements were satisfied through the development of DVI, MATLAB and dSPACE plug-ins respectively. The full list of stakeholder, functional and non-functional requirements of the PIT system and their implementation or traceability in the design is shown in Submission 6.

Number	Requirement	Type	How
STK_SRS_5	The system shall support configuration for monitoring and logging of in-vehicle network signals.	Stakeholder	Implemented under XML settings. Logging functionality has not been implemented for in-vehicle network signals not specified in the SDVM.
NFR_SRS_05	A MATLAB plugin shall support MATLAB version 2013b and 2014a.	Non-functional	Implemented in MATLAB 2013b. Not tested with MATLAB 2014a version.
NFR_SRS_19	The PIT system shall be designed to run on Windows XP, 7 and 8.	Non-functional	Tested with Windows XP and 7. Not tested with Windows 8 version.

Table 21. Partially satisfied PITS requirements.

5.2.6. Criterion #6 – Support of signal profiles

A set of 12 input/output signal profiles was introduced in Submission 3 and 4. These signal profiles are accessible within the SDVM for the definition of test cases. The PIT

system must be capable of supporting these signal profiles using the DVI engine for implementation in both offline and real-time test environments. Submission 6 shows the implementation of all 12 signal profiles in the SDVM. Most of the signal profiles are driven by a windows interface for easy and intuitive interaction with the user. The PITS application was evaluated against these signal profiles. The PITS application was able to generate and execute all 12 signal profiles on an offline MATLAB test environment. The real-time execution on dSPACE resulted in two signal profiles not being implemented successfully. The reason behind these two unsuccessful signal profiles lies with the current capability of the dSPACE system. The current version of dSPACE does not provide all input parameters required to implement these two specific signal profiles. A summary of the signal profiles implementation is shown in Table 37 in G.6. A signal profile sample (ramp signal profile) defined in SDVM and auto-generated by PITS through DVI is shown Figure 30. In Submission 6 a graphical representation of each auto-generation signal profile can be found including information about the auto-generated XML files.

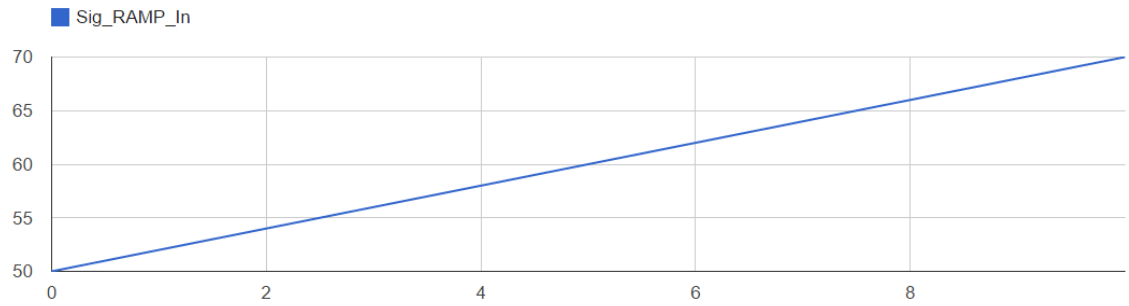


Figure 30. Ramp signal profile generated by PITS and driven by SDVM and DVI.

5.2.7. Criterion #7 – Support of offline and real-time automated testing

The proposed PIT system must support the execution of automated test scripts in offline and real-time test environments driven by test definitions in SDVM. The evaluation of this criterion was conducted using two vehicle use cases. The first vehicle

use case is the vehicle remote key locking control system whereas the second is the vehicle wiper control system. These use cases were chosen as they make good use of common automotive signal profiles such as single pulse (or on/off signal) and Pulse Width Modulation (PWM). In addition to that both use cases are easy to understand by the reader. Only one use case is presented in this report as both use cases produced the same test results. Full evaluation details and test results from both use cases are given in Submission 6.

The purpose of the vehicle remote key locking control system is to allow the user to lock and unlock the vehicle remotely using a key fob. A typical high level vehicle remote key locking system consists of five inputs and four outputs (see Figure 44 in Appendix G.7). The five inputs are: vehicle speed; crash signal; key fob lock button; key fob unlock button and multi/single entry configuration. Typical vehicle speed can range from 0 mph to 200 mph depending on a vehicle type. The crash signal is a binary signal with crash and no crash status. The key fob lock and unlock signals are binary with pressed and released status. Vehicle multi/single entry configuration signal is also binary with status single (only one door locked or unlocked) or multi (all doors locked or unlocked). The system outputs are the status of the four doors being locked or unlocked. Six test cases, shown in Table 22, derived from a typical driving scenario were considered for the evaluation of this criterion.

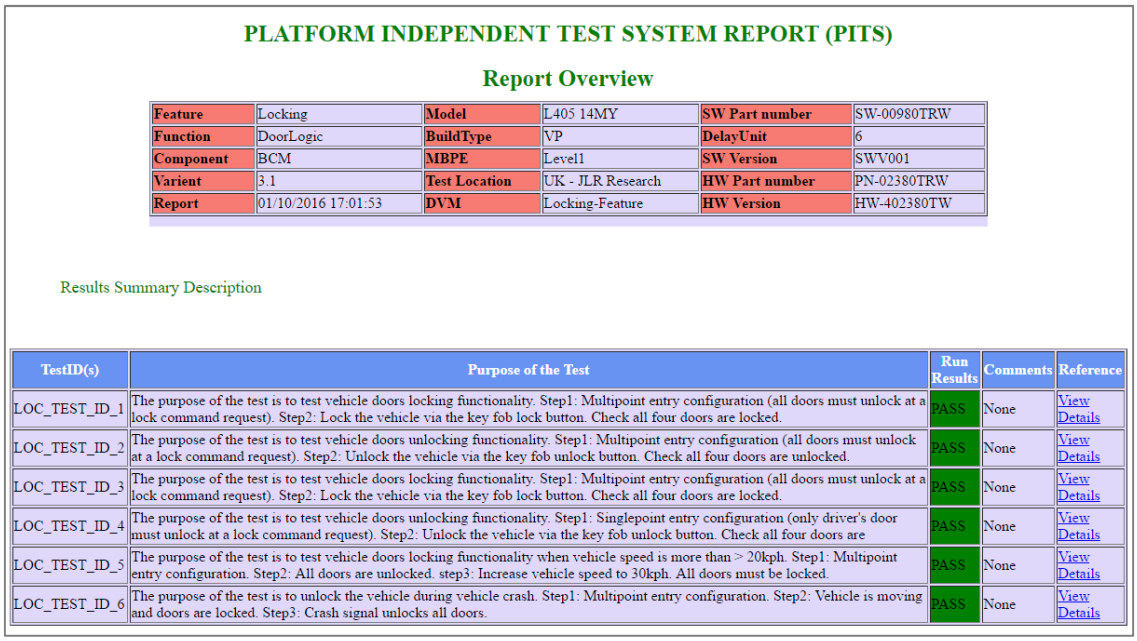
The test cases shown in Table 22 were transferred into SDVM. A sample of the test case definition in the SDVM is shown in Figure 45 in Appendix G.8. The PITS application was used in order to read the test definition in the SDVM and then auto-generate the test scripts suitable for automated execution in the MATLAB test environment. Prior to the execution of the automated test scripts, the PITS application

auto-generated a framework (or test harness) in order to create the interface between test scripts and the SIMULINK model.

		Test Case ID					
		LOC_TEST_ID_1	LOC_TEST_ID_2	LOC_TEST_ID_3	LOC_TEST_ID_4	LOC_TEST_ID_5	LOC_TEST_ID_6
Input Signals	Vehicle Speed	0 mph	0 mph	0 mph	0 mph	30 mph	30 mph
	Crash Signal	No Crash	No Crash	No Crash	No Crash	No Crash	Crash
	Lock	Pressed	Released	Pressed	Released	Released	Released
	Unlock	Released	Pressed	Released	Pressed	Released	Released
	Entry Config.	Multi	Multi	Multi	Single	Multi	Multi
Output Signals	Front Left Door	Locked	Unlocked	Locked	Unlocked	Locked	Unlocked
	Front Right Door	Locked	Unlocked	Locked	Locked	Locked	Unlocked
	Rear Left Door	Locked	Unlocked	Locked	Locked	Locked	Unlocked
	Rear Right Door	Locked	Unlocked	Locked	Locked	Locked	Unlocked

Table 22. High level test cases for vehicle remote key locking functionality.

Once the execution of the automated test scripts was complete the PITS generated a test report as shown in Figure 31. The number of test cases in Figure 31 matches the number of test cases in Table 22. Test cases ID and description is auto-generated from the SDVM test definition.



PLATFORM INDEPENDENT TEST SYSTEM REPORT (PITS)

Report Overview

Feature	Locking	**Model**	L405 14MY	**SW Part number**	SW-00980TRW
Function	DoorLogic	**BuildType**	VP	**DelayUnit**	6
Component	BCM	**MBPE**	Level1	**SW Version**	SWV001
Varient	3.1	**Test Location**	UK - JLR Research	**HW Part number**	PN-02380TRW
Report	01/10/2016 17:01:53	**DVM**	Locking-Feature	**HW Version**	HW-402380TW

Results Summary Description

TestID(s)	Purpose of the Test	Run Results	Comments	Reference
LOC_TEST_ID_1	The purpose of the test is to test vehicle doors locking functionality. Step1: Multipoint entry configuration (all doors must unlock at a lock command request). Step2: Lock the vehicle via the key fob lock button. Check all four doors are locked.	PASS	None	View Details
LOC_TEST_ID_2	The purpose of the test is to test vehicle doors unlocking functionality. Step1: Multipoint entry configuration (all doors must unlock at a lock command request). Step2: Unlock the vehicle via the key fob unlock button. Check all four doors are unlocked.	PASS	None	View Details
LOC_TEST_ID_3	The purpose of the test is to test vehicle doors locking functionality. Step1: Multipoint entry configuration (all doors must unlock at a lock command request). Step2: Lock the vehicle via the key fob lock button. Check all four doors are locked.	PASS	None	View Details
LOC_TEST_ID_4	The purpose of the test is to test vehicle doors unlocking functionality. Step1: Singlepoint entry configuration (only driver's door must unlock at a lock command request). Step2: Unlock the vehicle via the key fob unlock button. Check all four doors are	PASS	None	View Details
LOC_TEST_ID_5	The purpose of the test is to test vehicle doors locking functionality when vehicle speed is more than > 20kph. Step1: Multipoint entry configuration. Step2: All doors are unlocked. step3: Increase vehicle speed to 30kph. All doors must be locked.	PASS	None	View Details
LOC_TEST_ID_6	The purpose of the test is to unlock the vehicle during vehicle crash. Step1: Multipoint entry configuration. Step2: Vehicle is moving and doors are locked. Step3: Crash signal unlocks all doors.	PASS	None	View Details

Figure 31. Offline automated testing, test results for vehicle remote key locking control system.

The user can view each test case separately by selecting *View Details* under the reference column. A sample of test results, in this case LOC_TEST_ID_2, is shown in Figure 46 and Figure 47 in Appendix G.9. The user can easily assess and establish the pass status of the actual output signals. The automated test results report is driven by HTML and XML files. This means that test results are tool independent and hence can be shared with other key project stakeholders. The PITS application together with the SDVM and DVI was able to successfully extract and execute all test cases in the MATLAB test environment.

The evaluation then moved into real-time using dSPACE hardware as the test target. The evaluation was conducted using the same test cases previously used for the offline testing. The execution of the automated test scripts was conducted on a dSPACE MicroAutobox II DS1401 real-time processor. The same vehicle remote key locking control system SIMULINK model was compiled in order to generate C code suitable for

real-time execution. A real-time interface called ControlDesk was developed as shown in Figure 32, in order to assess the correctness of the auto-generated C code functionality in the real-time target. The user can operate the Graphical User Interface (GUI) in order to change any of the inputs and observe the behaviour or state of the outputs. In Submission 6 more ControlDesk views for the vehicle remote key locking control system are given.

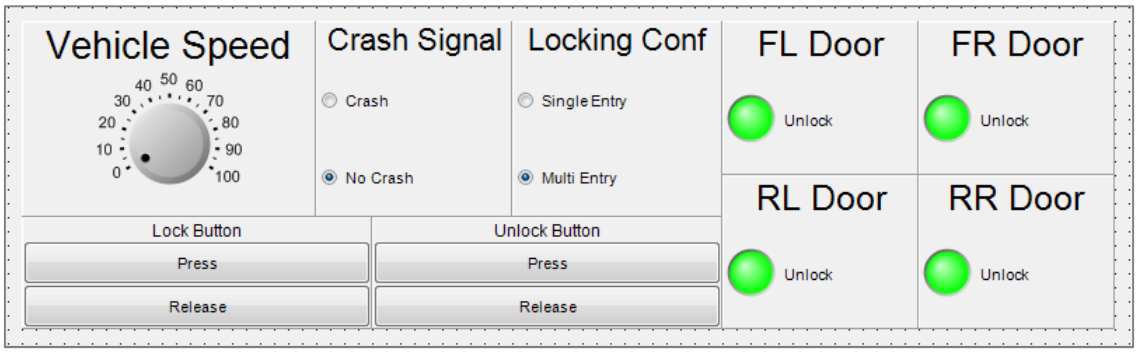


Figure 32. dSPACE ControlDesk interface for vehicle remote key locking control system.

Once the correctness of the C code in real-time was established the next step in the evaluation was to execute the automated test scripts. Real-time test results for test case LOC_TEST_ID_2 are shown in Figure 48 and Figure 49 in Appendix G.10. Comparing the offline and real-time test results for the same use case and test case, it can be concluded that the outcome of the test is the same. The actual output signals response falls within the maximum and minimum tolerance values. A minor timing deviation can be observed by looking closely at the actual output signals from both offline and real-time test results. This is due to the synchronisation between the real-time target and the offline environment that the automated test script is executed. The timing discrepancy between the two results can be eliminated with further development of the PITS application. This is outside the scope of this research.

5.2.8. Criterion #8 – Capability to support all MBPE levels

The proposed PIT system must be able to auto-generate and execute test scripts from test cases derived across all Model-based Product Engineering (MBPE) process levels. In Section 3, it was stated that one of the challenges within the automotive industry is the lack of end-to-end solutions to support embedded software development and test across all levels of abstraction. This is a key enabler to shift failure modes and/or software defects to the early stages of the product development process. In order to evaluate how capable is PITS of supporting all levels of abstraction a benchmarking assessment against other commonly used Commercial-off-the-Shelf Tools (COTS) was conducted using the Pugh Matrix approach (Pugh, 1991), (Pugh, 2009). The Pugh Matrix approach allows comparison of a number of design candidates leading ultimately to which best meets a set of criteria. The Pugh Matric approach is easy to use and this is the main reason of its popularity within the automotive and other industries (Fernandes *et al.*, 2008), (Cervone, 2009), (Thakker *et al.*, 2009), (Thorén and Burgren, 2015), (Harris *et al.*, 2016). The key criteria used in this benchmarking assessment were the six MBPE levels. The result is shown in Figure 33. The names of the COTS have been removed due to JLR's confidentiality. PITS support for all MBPE levels produced high scores and as a result PITS outperformed all six COTS tools. The benchmarking assessment was conducted with a group of experienced test engineers within JLR being familiar with the chosen COTS tools. It is important though to state that each of the chosen COTS tools is capable and suitable for certain engineering activities. Most tool vendors chose a specific part of the automotive development cycle to develop tools and as a result in most cases all COTS have their own interface. This creates a challenge for the automotive OEMs as very often they decide to change tool sets, hence resulting in huge amount of rework to re-generate test cases from one tool

to another. The proposed PIT system closes that gap. Full details related to this benchmarking assessment are provided in Submission 6.

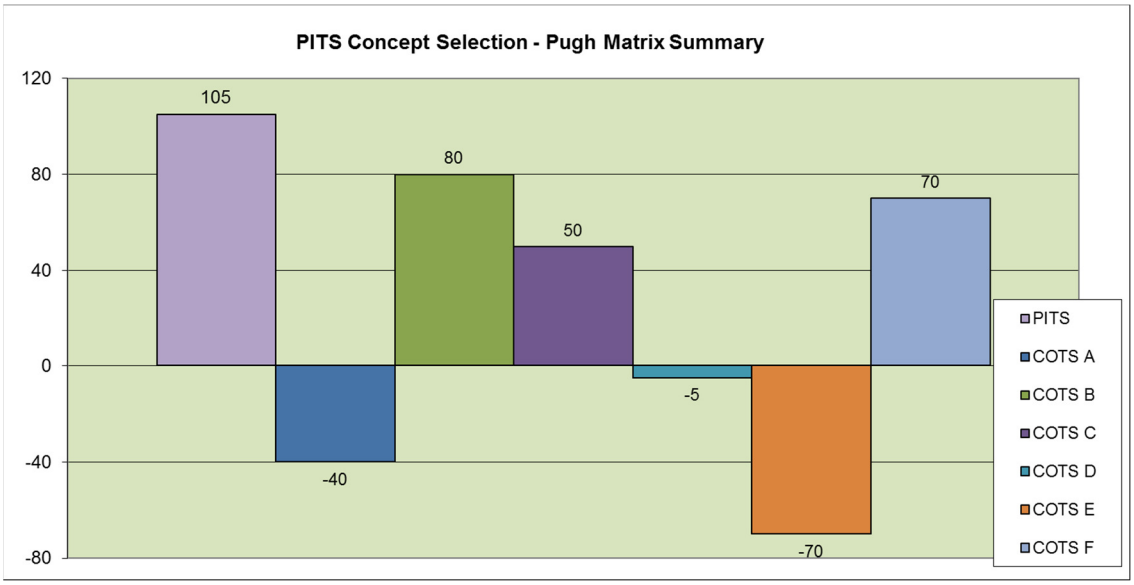


Figure 33. Graphical summary of Pugh matrix results– PITS concept selection against MBPE levels and other COTS.

5.2.9. Criterion #9 – Efficiency in terms of engineering effort and time

Efficiency in terms of engineering effort and time has been a subject for research for a number of years (Chaudhary and Yadav, 2012), (Hanna *et al.*, 2013), (Sharma, 2014), (Kumar and Mishra, 2016). Many industries are seeking new methods, tools and processes to reduce engineering effort and time to market. It is therefore imperative to evaluate PITS against engineering effort and time.

The data for this case study were derived from an existing JLR vehicle programme. A vehicle feature DVM consisting of approximately 2,500 test cases was considered for the evaluation. The engineering effort for both manual and automated testing has been split into creation and execution of test cases. The creation involves the engineering activity to generate test definitions for either manual or automated testing. The

execution involves the engineering activity to run test cases using either manual or automated testing. Figure 34 depicts the engineering effort for manual and automated testing using COTS tools and PITS for the first embedded software test iteration only.

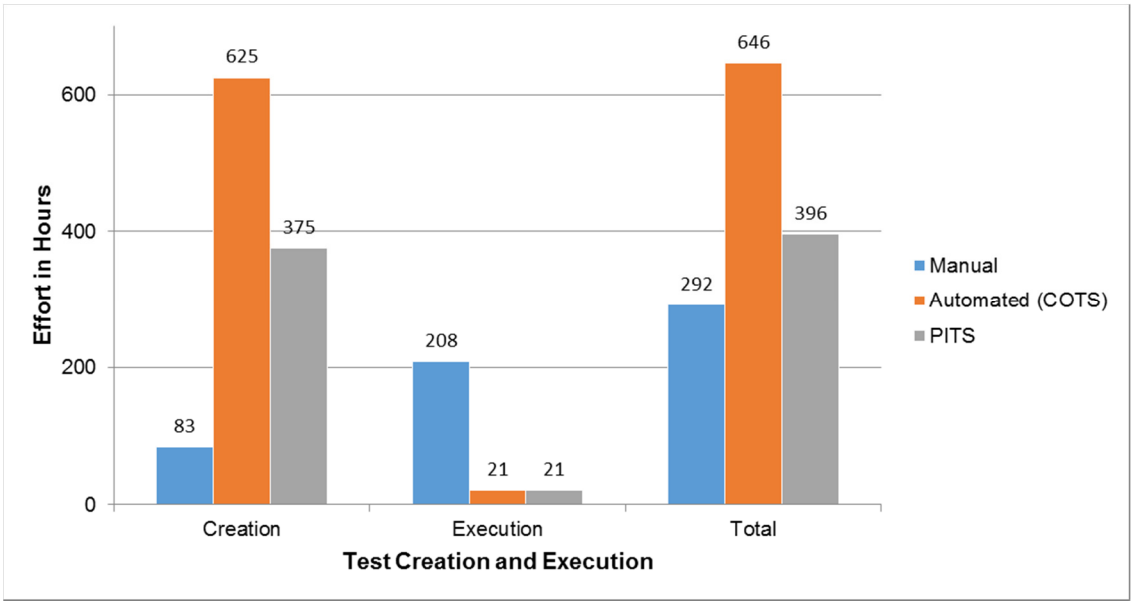


Figure 34. Comparison of manual and automated testing using COTS and PITS. Effort for test creation and execution for the first iteration.

As can be seen from Figure 34 the engineering effort to create and execute test scripts suitable for automated testing is significantly greater than the engineering effort to create and execute test cases manually. This is due to the fact that creating test scripts is time consuming engineering activity. The engineering effort for the automated testing depicted in Figure 34 has been calculated as the average time taken to create and execute test scrips using the previous list of COTS tools and PITS.

The engineering effort for execution is the same between PITS and COTS. This is an expected outcome as both solutions depend on a given test target. The main difference and benefits of PITS is related to the engineering effort to create test scripts. The integration of PITS with the SDVM and DVI results in faster creation of test definition

suitable for automated testing. The engineering effort to create test scripts has been reduced by approximately 40%.

As shown in Figure 35 this reduces significantly the number of software iterations for breakeven between manual and automated testing to less than two from three using COTS tools. In terms of engineering time rather than effort if only 28 weeks were available to create a product, the use of PITS (together with SDVM and DVI) can accommodate 36 embedded software test iterations compared to 24 using COTS and 5 using manual testing. There are additional benefits with PITS that were not considered in the calculations such as effort for test scripts update and change of test targets or test platforms. More evaluation details can be found in Submission 6.

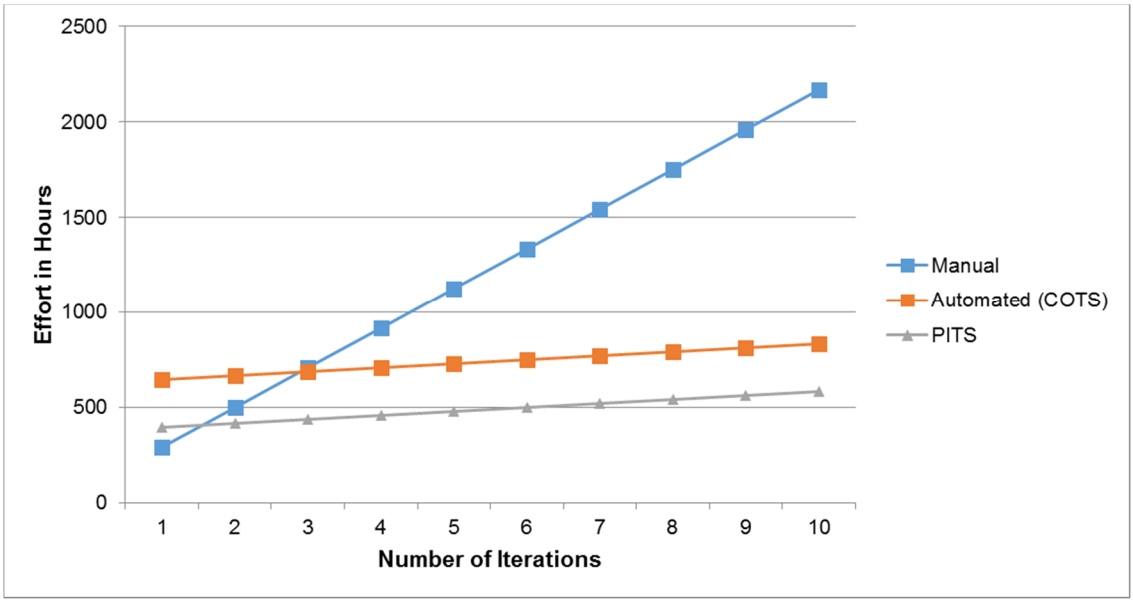


Figure 35. Comparison of manual and automated testing using COTS and PITS. Effort for test creation and execution for 10 iterations.

5.2.10. Criterion #10 – Sustainable solution

The main question around sustainability is as follows:

Question: Is the PITS application together with the SDVM, DVI and MBPE process sustainable to future toolsets changes?

The DVI part of PITS is generic and platform independent. If in the future test automation tools change then only a new plugin is required in order to keep the PIT application operational. The SDVM is subject to potential changes if the company decides to go for a COTS solution in the future. In this case the format of the proposed SDVM will be used to generate a set of key requirements and best practice for creating future DVMs using a semi-formal test specification approach. The PITS application will require a new extractor to be developed in order to translate test cases written in future toolsets to DVI interface. The DVI interface provides long term sustainability as test scripts can be archived for future use with new toolsets. It is also important to mention here that the nature of the MBPE process is generic and therefore sustainable to future changes.

5.2.11. Criterion #12 – Early detection

A key requirement of the MBPE process is the design, verification and validation of automotive embedded software prior to supplier activities. Software verification and validation prior to supplier activities has the potential to drive early detection of failure modes and/or software defects in the product development process. In Submission 2 two cases studies were presented in order to demonstrate the effectiveness of the proposed process. Both case studies targeted the early levels of the proposed process in order to prove that detection of failure modes and/or software defects can be shifted

to the early stages of the product development. More details on both case studies can be found in Submission 2.

The first case study focused on the development of the park assist vehicle feature at the requirements phase of the design and particularly the engineering activities defined in MBPE level 0. As a result engineering activities defined in Level 0 were conducted which lead to the creation of full requirements engineering analysis and proof of concept creation as shown in Figure 36.

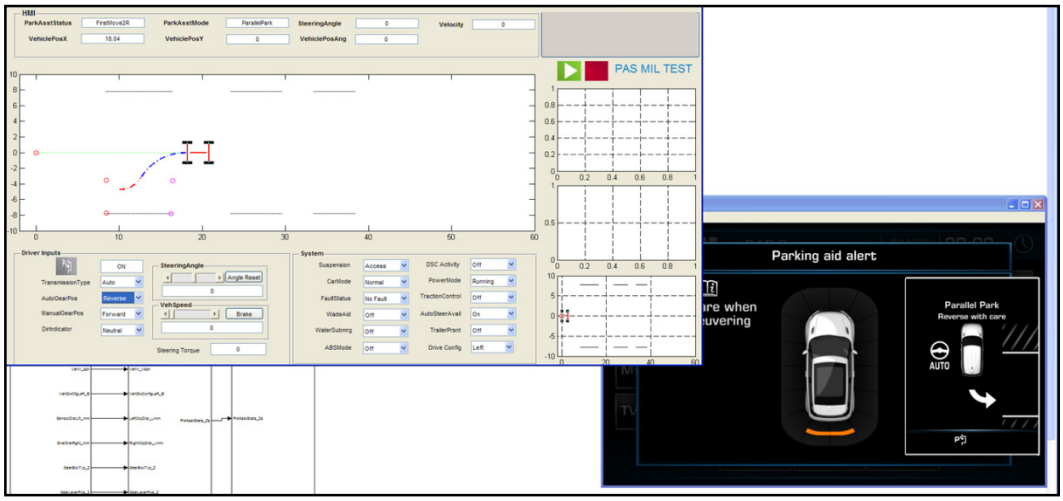


Figure 36. Proof of concept development for park assist vehicle feature.

The outcome of the engineering activities in Level 0 resulted in early detection of failure modes and/or software defects for the park assist vehicle feature as shown in Table 23. A number of requirements were found to be missing from the initial set of requirements or were too ambiguous for implementation. In addition to that, a new set of requirements were found for the completeness of the intent feature functionality. All these requirements would have been found later in the product development through embedded software testing and would have caused late and expensive changes. This is a substantial improvement compared to the existing engineering practice within the

company. In the previous vehicle programmes written requirements were analysed manually and then sent to the suppliers. The introduction of Proof-of-Concept (PoC) development and Requirements Engineering (RE) has helped to generate better and more robust embedded software requirements with fewer potential failure models and/or software defects.

Failure Mode and/or Software Defect	% of Initial Number of Requirements
Missing requirements	6%
Ambiguous requirements	5%
New requirements to improve functionality	11%

Table 23. Evaluation results from park assist vehicle feature.

The second case study focused on the development of body electronics vehicle features such as remote locking, alarm, passive entry and passive start. Body electronics vehicle features are distributed across vehicle networks and have high level of interaction with the user (vehicle driver and passengers). This case study targeted the early levels of the MBPE process in order to assess the effectiveness of Levels 0, 1 and 2. Implementation models for body electronics were developed in MATLAB and were used for both manual and automated testing prior to supplier software release. The duration of this case study was approximately 10 months. During this period the majority of the engineering activities defined in Level 0, 1 and 2 were executed for the evaluation of the process. The duration of this study did not include the development time for the implementation models. The results[3] of the evaluation are shown in Figure 37. Over 50% of the failure models and/or software defects were found prior to supplier software release. This is a significant improvement compared to the previous

[3] Due to vehicle programme variations and changes and the introduction of many variables it is not appropriate to depict the results in Figure 37 in a similar programme development gateway timeline as was shown in Figure 4 in Section 1.4.3.

programmes where failure models and/or software defects were typically found after supplies' software release. Functional (or implementation) models were validated using MIL and RCP testing. The models' validation included integration testing in a MIL environment with models from other vehicle systems and features. The RCP testing included integration testing with carry-over ECUs (legacy systems) from other systems or functional models developed for new systems and features.

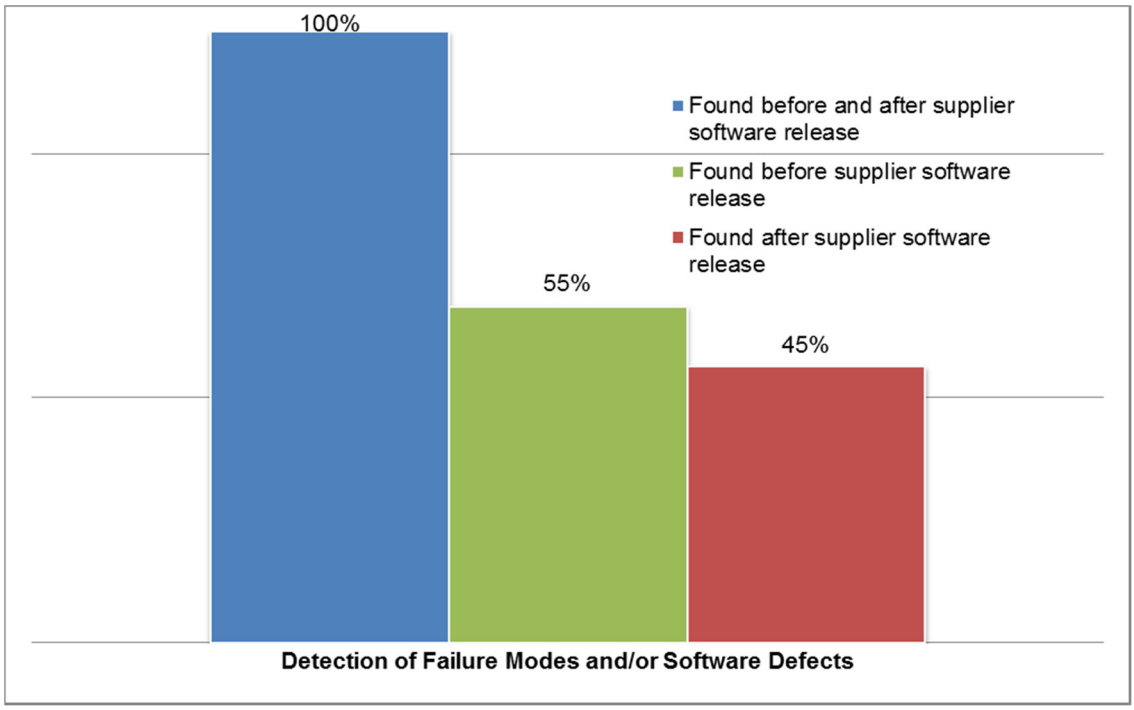


Figure 37. Evaluation results from body electronics.

The result from this case study is a tangible demonstration of the MBPE process effectiveness on automotive embedded software quality. During the evaluation period engineers where able to iterate and loop amongst Levels 0, 1 and 3 until the correct implementation and functionality was proven. An interesting outcome from this evaluation is the type of failure modes and/or software defects which were found before and after supplier software release and availability of production ECU hardware. A summary of failure modes and/or software defects and the detection capability of the

MBPE process at each Level is shown in Table 32 in Appendix G.2. Some failure modes and/or software defects are coupled with low level embedded software and availability of production hardware.

5.2.12.Criterion #12 – Low cost

The low cost evaluation criterion assesses the total cost to the company to purchase and maintain COTS against the PITS application. Consider l as the number of licences needed over a given period in years defined by y. If the cost of license purchase is defined by c and the annual maintenance cost as a percentage per license is defined by mp, and the actual annual maintenance cost per license is defined by mc then the total cost to the company, tc, is given by:

$$mc = mp * c,\ tc = (l * c) + (mc * l * y)$$

A typical use case for JLR is shown below:

$$l = 100,\ y = 5,\ c = £7{,}000,\ m = 20\%$$

$$tc = (l * c) + (c * m * l * y) = £1{,}400{,}000$$

The cost of £7,000 per license was taken as the average cost amongst the list of COTS tools considered in the previous sections. The 20% annual cost for maintenance is a typical cost across almost all tool vendors. The expense of £1.4M is a significant cost to the company for 100 licences (an approximation derived from previous vehicle programmes) over a five year period. Figure 38 gives a cost forecast of PITS against COTS and the potential savings to the company. The annual maintenance cost of PITS was estimated £60K compared to £140K of COTS. 60K annual maintenance cost cover

the support for one full-time engineer dedicated to maintain and improve the functionality of PITS. The annual maintenance cost was an approximation since engineering rates are subject to changes over a period of time. As a result significant savings result from the non-purchase of COTS (the application of PITS can be shared with any engineer within JLR). The potential savings to the company are substantial and initially in the range of £780K resulting in total savings over a five year period of £1.1M.

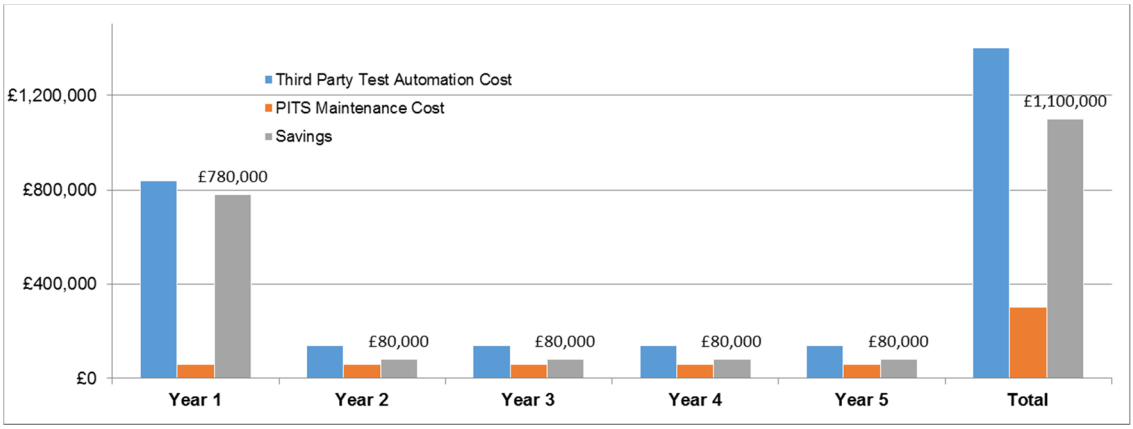


Figure 38. Forecast of accumulate total savings resulting from the proposed PIT system over five-year period.

5.2.13. Criterion #13 – Low maintenance

The tool maintenance evaluation criterion excludes license and maintenance cost. The primary aim of the tool maintenance evaluation criterion was positioned around changeability, accessibility (or availability) and learnability. Changeability is referring to potential changes and updates being requested by test engineers. As the source code of the PITS is own by JLR, changes and updates of PITS can be delivered at short notice without the need to raise purchase orders to COTS vendors. Accessibility refers to a software licence or application being accessible by all engineers and JLR suppliers at all time any time. Very often companies restrict the use of certain software due to limited number of licenses available. The PITS application is accessible to everyone

within JLR and potentially across JLR's supplier base. Learnability is a key challenge for many companies as very often they have to make significant investment on training for new skills. Fragmentation of COTS within companies drives significant training costs and loss of engineering time. The PITS application reduces the need for specialised skills. It offers a simple user interface which is linked to SDVM. The SDVM is Microsoft Excel-based. Microsoft Excel is a tool that most engineers are familiar with and can access within their own departments

5.2.14. Criterion #14 – Overall comparison

In order to fully complete the evaluation of PITS and provide more rounded picture about benefits and impact to the company a selection of key evaluation criteria from Table 19 were used to assess PITS against the set of COTS tools from the previous sections. The assessment was conducted using the Pugh Matrix approach described in the earlier sub-sections. Some evaluation criteria were excluded from the assessment as these criteria are PITS specific and cannot be used for comparative analysis. The reduced number of selected key criteria which qualified for the analysis is shown in Table 24.

Key Criteria	Weight (%)
Standardisation	15
Portability	12
Offline Support	12
Real-Time Support	13
MBPE Levels Support	8
Efficiency	14
Sustainability	9
Cost	11
Tool Maintenance	6

Table 24. Reduced number of evaluation criteria.

A graphical representation of the Pugh Matrix assessment is depicted in Figure 39. The results look different from the previous Pugh Matrix assessment against the MBPE process levels evaluation criteria.

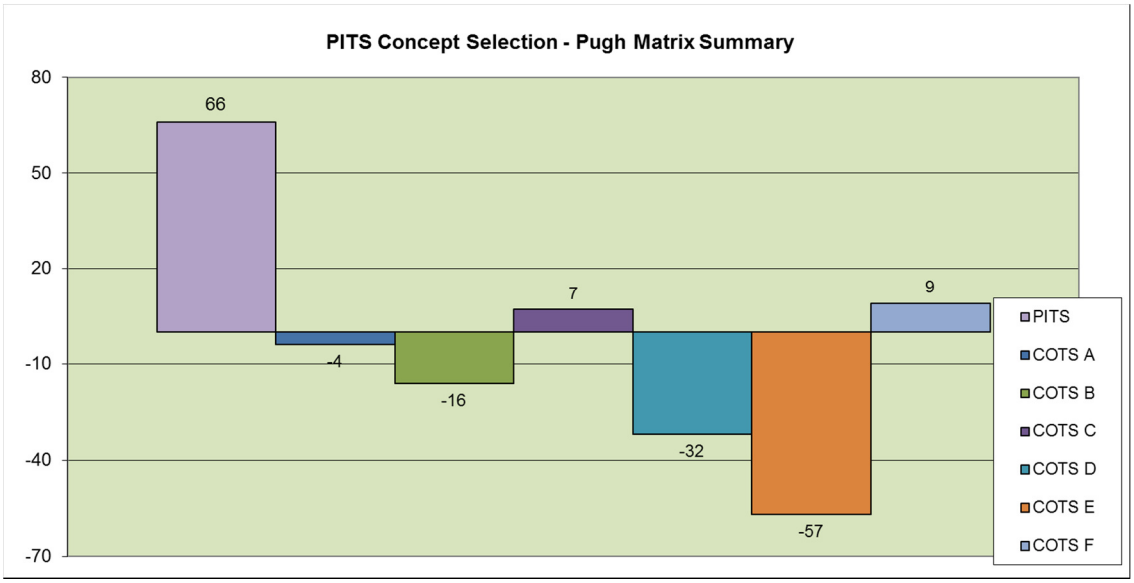


Figure 39. Graphical summary of Pugh matrix results– PITS concept selection against COTS tools using a selection of key evaluation criteria.

The portability and efficiency strength of COTS F changes the ranking. COTS C scores were elevated from fourth to third. This does not mean necessarily that COTS F is better tool than COTS B or COTS C. It means that taking a holistic approach that ranges from standardisation to maintenance and cost COTS F is potentially the preferred COTS tool after PITS. If however an application needed strong real-time support then a COTS B solution would have been preferable compared to COTS F and PITS. In any case the assessment and analysis shows strong preference for PITS. The integration of PITS with the MBPE process, the SDVM and DVI drives high scores in the analysis. Full details of the analysis are shown in Submission 6.

5.3. Summary of evaluation results and concluding remarks

A summary of the evaluation results is shown in Table 25. Similar to the requirements definition phase, internal JLR stakeholders such as test engineers, model developers and senior management, were involved in the final review of the evaluation results. Concluding remarks can be drawn as follows:

- Detection of failure modes and/or software defects of over 50% prior to supplier embedded software release. This is a significant improvement compared to previous vehicle programmes where failure modes and/or software defects were found mainly after suppliers' software release. The improvement included early detection of failure modes and/or software defects associated with the functional (or implementation) models and requirements for embedded software.
- End to end solution that supports all levels of abstraction from the test case definition phase to the execution of automated test scripts in both offline and real time test environments.
- 40% engineering effort reduction for test script creation for automated testing.
- Five to tenfold engineering time reduction compared to manual and automated testing using COTS tools.
- Sustainable solution for maintaining and archiving test scripts for future use using future COTS toolset.
- Cost reduction for licence and maintenance in the range of £1.1M over a five year period.
- Solutions are generic and tool independent hence applicable beyond JLR and the automotive industry.

Number	Criterion	Satisfied	How/What
1	Satisfy MBPE process requirements listed in Table 9	Y	All requirements were **satisfied**. See Table 31 in Appendix G.1.
2	Satisfy DVI requirements listed in Table 13	Y	All requirements were **satisfied**. See Table 33 and Table 34 in Appendix G.3.
3	Satisfy SDVM requirements listed in Table 14	Y	All requirements were **satisfied**. See Table 20.
4	Test target (or platform) independent	Y	A test definition in SDVM was used to generate and execute automated test scripts in two **different test targets** (see Figure 29).
5	Satisfy PITS requirements (stakeholder, functional and non-functional)	Y	All PITS requirements were **satisfied** apart from one stakeholder and two non-functional which they were partially satisfied mainly due to software availability constraints. A sample is available in Table 36 in Appendix G.5.
6	Support of different signal profiles as defined in the DVI and SDVM	Y	Signal profiles were **successfully implemented** on both MATLAB offline and dSPACE real-time test environments (see Table 37 in Appendix G.6). Two signal profiles were not implemented in real-time due to dSPACE test environment **constraints**.
7	Support of offline and real-time automated testing	Y	The PITS application **successfully** auto-generated test scripts defined in SDVM and executed them in both offline and real-time test environments (see Figure 46 and Figure 47 in Appendix G.9 and Figure 48 and Figure 49 in Appendix G.10).
8	Capability to support all MBPE levels	Y	Results from benchmarking assessment against other COTS tools have shown that PITS has **better capability** to support all MBPE levels.
9	Sustainable solution	P	DVI and MBPE are **sustainable**. PITS and SDVM are subject to future changes.
10	Early detection of failure modes and/or software defects	Y	Deployment on body electronics. Detection and shift of failure modes and/or software defects of **over 50%** prior to supplier software release (see Figure 37). Deployment on park assist feature. Evidence to suggest that Level 0 of the MBPE process can **significantly improve** the quality of software requirements (see Table 23).
11	Efficiency in terms of engineering effort and time	Y	**40% engineering effort reduction** for test script creation suitable for automated testing. Engineering time reduction compared to COTS and manual testing.
12	Low cost	Y	Potential **savings** to the company are substantial and initially in the range of £780K resulting in total savings over a five year period of £1.1M
13	Low maintenance	Y	**Reduces** the need for specialised skills. Source code of PITS application is owned by JLR.
14	Overall comparison against COTS tools	Y	Results from benchmarking assessment against other COTS tools have shown that PITS is a **better end-to-end solution**.

Table 25. Summary of evaluation results.

6. Key innovations, benefits, lessons learned and future work

This section presents the main claim of innovation and how it is driven by key solutions that have resulted from this research and development project. The benefits of this research to Jaguar Land Rover (JLR) and its supplier base are discussed and summarised. This section concludes with lessons learned and a set of recommendations for future research.

6.1. Key innovations

In Section 2 it was stated that the aim of this research project was to research and develop solutions in order to shift failure modes and/or software defects detection to earlier phases of the automotive product development process.

The innovation claim for this research project is stated as follows:

Solutions for embedded software that shift failure modes and/or software defects detection to earlier phases of the automotive product development process.

The claim is driven by four key solutions. Each solution addresses a specific gap and need within the automotive industry and as a result has its own benefits. ***The integration of all four key innovative solutions makes the holistic research outcome more than the sum of its parts***. The four key solutions are:

- **Model-based product engineering process**

  The first solution that has resulted from this research is the creation of a new process for automotive embedded software development called Model-based Product Engineering (MBPE). Submission 2 and Section 4 have shown that

previous research studies and current literature indicate that there is a need in the automotive industry for new integrated standardised model-based processes for embedded software development. The MBPE process addresses this gap. The MBPE process is standardised and tool independent. It consists of six levels with clear set of deliverables that are well understood by engineers and managers as well as automotive suppliers. The research has resulted in the creation of new engineering standards and design rules. In addition to that the MBPE process addresses the need to support product complexity assessment for seamless deployment within JLR and its supplier base. The deployment of the MBPE process within JLR has enabled detection of failure modes and/or software defects of over 50% prior to supplier embedded software release.

- **Generic design verification interface**

  The second solution that has resulted from this research is the creation of a new generic Design Verification Interface (DVI). The new generic DVI allows vehicle test cases to be exchanged and shared amongst JLR departments and suppliers. These test cases are sustainable for future use due to reduction of redundant engineering effort and time to re-use existing test cases on different test tools and test targets. In addition to that the generic DVI offers test case traceability across all MBPE levels and an interface where test cases can be used for automated testing in both offline and real-time test environments. Submission 3 and Section 4 show the research findings and evidence confirming the need for this innovative solution.

- **Standardised design verification method**

  The third solution that has resulted from this research is the creation of a new Standardised Design Verification Method (SDVM). In Submission 4 and Section 4 it

was shown that currently within JLR and the automotive industry test specifications are written in a completely different format hence making it very difficult for test engineers to auto-generate test scripts directly from these documents. The heterogeneous format of test specifications leads to significant re-work and re-engineering of the software as many tests are implemented incorrectly hence causing late software changes resulting in loss of engineering effort and time. The SDVM addresses this gap. The SDVM offers a common and standardised interface for defining test cases derived from different vehicle systems. The semi-formal specification approach of the SDVM allows test scripts to be auto-generated directly from test specifications.

- **Platform independent test system**

  The fourth solution that has resulted from this research is the creation of a new Platform Independent Test System (PITS). In Submission 4 and Section 4 it was shown that the transition from manual to automated testing involves the creation of test cases and test scripts that are suitable for execution in both offline and real-time-time test environments. Current best practice drives the creation of test cases and test scripts to be developed for specific test tools and test targets. Test cases and test scripts developed for specific test targets require significant engineering effort and time due to the creation of rework and lack of test cases re-use. The holistic design and development of a new Platform Independent Test System (PITS) addresses this gap. The PIT system provides an end-to-end solution that supports automation across all MBPE process levels. The evaluation results of PITS have concluded research findings of 40% engineering effort reduction for test script creation and five to tenfold engineering time reduction compared to manual and automated testing using Commercial-off-the-Shelf (COTS) tools. The PITS solution reduces license and maintenance costs for test automation tools. In

addition to that, provides JLR and its supplier base with a test environment which is easy to upskill engineers and promotes effective and efficient engineering resource utilisation.

6.2. Benefits

A summary of the benefits to JLR and its supplier base resulting from the solution developed in this research is shown in Table 26.

The research outcome of this project has enabled a shift of failure modes and/or software defects detection to earlier phases of the automotive product development process. JLR and its supplier base can develop future vehicle embedded software much faster with improved quality. This is driven by reduced engineering effort and time required to engineer and test future products mainly vehicle systems and features.

Due to customer expectations for new and innovative features, the automotive industry and JLR will continue to face the challenge of managing the increased complexity of embedded software in vehicles. The innovative solutions developed throughout this research project will facilitate JLR and its supplier base with new processes, methods and tools capable to address this challenge.

The use of the solutions within JLR and its supplier base will accelerate the roll-out of new vehicle features and functions. JLR is now in a stronger and more competitive position to develop in-vehicle embedded software. The quality of the embedded software can significantly improve due to the early introduction of validation and verification engineering activities prior to supplier software release.

Innovative Solutions	Key Benefits
Model-based Product Engineering (MPBE)	• Standardised and tool independent process for embedded software development. Successfully deployed within JLR and its supplier base. • Six levels with clear set of deliverables that cover all key stages of the embedded software development driven by engineering standards and design rules. • Integrated deployment strategy that supports product complexity for robust introduction within a large automotive organisation. • Integration and link to JLR's internal product development process. • Three levels of verification and validation (V&V) before Electronic Control Unit (ECU) prototype is available. • Embedded best practice from model-based development and systems engineering and other key product development processes. • Detection of failure modes and/or software defects of over 50% prior to supplier embedded software release.
Generic Design Verification Interface (DVI)	• Generic test interface suitable for exchanging and sharing test scripts amongst JLR and its supplier base. • Traceability across all MBPE levels. • Sustainable solution for maintaining and archiving test cases for future use. Reduces redundant engineering effort and time to re-use existing test cases on different test tools and test targets.
Standardised Design Verification Method (DVM)	• Reduction of engineering effort and time to develop DVMs suitable for automated testing. • Easier to cope with late DVM changes and test script updates due to auto-generation of test scripts decoupled from test tools and test targets. • Common and standardised interface for defining test cases derived from different vehicle systems. • Automatic extraction of automated test scripts suitable for execution in both offline and real-time test environments.
Platform Independent Test System (PITS)	• End-to-end solution that supports automation across all MBPE process levels. • Support for offline and real-time automated testing enabling early testing before vehicle prototypes and production ECUs are available. • 40% engineering effort reduction for test script creation suitable for automated testing. • Five to tenfold engineering time reduction compared to manual and automated testing using Commercial off the Shelf (COTS) tools. • Reduction in license and maintenance costs for test automation tools. • Easy to upskill engineers and effective resource utilisation across the company.

Table 26. Key benefits of innovative solutions.

Requirements Engineering (RE) coupled with early Proof-of-Concept (PoC) development and Model-Based Development (MBD) drives early detection of failure modes and/or software defects. Requirements specification is more complete and where appropriate in-house developed implementation/functional models can be used for auto-coding speeding up the development of new and innovative vehicle features. Engineers and managers have a new holistic framework in place that allows them to assess much faster what resources and engineering skills are needed to fast track the development of new and complex vehicle features.

The lack of automation at the early stages of the product development process has been addressed in this research. JLR and supplier base have now the capability to execute automated tests prior to vehicle prototypes and Electronic Control Unit (ECU) hardware availability. Test specifications can be written in a SDVM for auto-generation of test scripts. The variation of having different vehicle systems and features DVMs written in different test specification forms has been removed since the introduction of the proposed SDVM. New test specifications can be used for both manual and automated testing. Test cases can be archived for future use using future test tools and test targets (or test platforms). It is difficult to accurately estimate the cost of re-writing test cases suitable for different test tools and test targets. The cost though is significantly high for both the OEM and its supplier base. This cost has been substantially reduced. JLR has the opportunity in the future to change test automation tools without having to occur additional and unnecessary costs for re-writing test scripts from existing test specifications. Training for new test tools suitable for automated testing can also substantially reduced. The proposed PIT system is driven by a simple Windows user interface. The associated user interface complexity from Commercial-off-the-Shelf (COTS) tools has been reduced. Test engineers no longer required to

acquire programming and computer science engineering skills in order to operate test tools that are often complex with non-intuitive user interface.

The evaluation results where appropriate have demonstrated the benefits of the innovative solutions through concrete evidence driven by engineering data. Data have been generated from real world use cases and business needs. Key benefits such as detection of 50% of failure modes and/or software defects prior to prototypes and ECU hardware availability, 40% reduction in engineering effort for test scripts creation, five to tenfold reduction in engineering time for automated testing, clearly demonstrates the significance of this research project to JLR and its supplier base.

6.3. Lessons learned

Although the proposed solutions have provided key benefits to the company and its suppliers, the research project has also provided vital lessons which have to be taken into consideration.

The deployment of the solutions within JLR and its suppliers has exposed a number of challenges. The automotive industry is still in its infancy in terms of engineering skills availability for undertaking engineering related activities such as requirements engineering and model-based development. During this research it was found that a considerable amount of training and time was required for automotive engineers, very often with little software development experience, to become proficient and competent in toolsets aiming to be used for requirements engineering and model-based development. Lack of skills was particularly identified in the case where functional (or implementation) models intended to be used to automatically generate production software. It was also noted the need for up-front investment associated with the

engineering effort to develop Proof of Concepts (PoCs) using abstract models. Engineering through PoCs drives fast-fail mind-set and requirements churn at the early phases of the product development process. This is particularly challenging in a dynamic environment where numerous other vehicle programmes are being run in parallel.

Another notable significant challenge was the commitment for capital expenditure for the provision of simulation hardware and software. The proposed innovative solutions drive a set of simulation tools (both hardware and software) that often are more expensive in comparison with conventional software and test development methods. This challenge was addressed with the introduction of a deployment strategy suitable to determine the applicability of simulation tools at each stage of the embedded software development.

The verification and validation engineering activities of the proposed solutions necessitated the need for sensor, actuator and plant simulation models being developed for both offline and real-time automated testing. In some cases plant models were too detailed and extremely complex which led to substantial engineering effort for model debugging and integration with chosen simulation tools. A key lesson learned here is that model-based development experience is essential and required in order to judge the level of fidelity of the model under consideration. If a sensor, actuator or plant model is too abstract then it might be insufficient for integration. On the other hand if a model is too detailed then as mentioned earlier integration challenges arise.

6.4. Future work

The completion of this research project has presented opportunities for future work in the area of automotive embedded software development.

Currently test scripts are generated from test cases defined and written by system engineers. These test cases do not necessarily take into account all levels of system abstraction. Because test cases are written using a manual procedure they tend to cover systems and features that are at the higher level of abstraction. This is driven from the nature of automotive systems being highly interconnected and coupled. The consequence can lead to the conclusions that the full test space and coverage of a given system or feature under test remains unexplored. The lack of systematically deriving test cases at all levels of abstraction can lead to undiscovered failure models and/or software defects.

Another opportunity for future research work is around the development and engineering of requirements and test specification. System requirements and test specifications tend to look for “passes” rather than “fails”. Fail conditions within test specifications are driven from non-functional requirements as well as requirements generated through other techniques such as pairwise, combinatorial and random testing. The robustness of the automotive embedded software can be substantially improved if test specifications were “intelligent” to look for unintentional functionality in the system under consideration.

The solution of the SDVM can be extended to cover requirements specification. If requirements were to be written in a standardised form then the auto-generation of test scripts can be more effective and productive. Current toolsets available to the

automotive industry allow requirements specification to be written in a machine readable data form but they still leave it open to the engineers to customise and create bespoke environments, very often driving inconsistency across the OEMs and suppliers.

The trend of vehicle embedded software being moved to an off-board server (or cloud) necessitates the need for new research in the area of off-board vehicle feature development and test. Do off-board vehicle software engineering activities need to follow best practices developed for on-board? What are the key enablers for fast pace development and introduction of new and innovative off-board based vehicle features? How do these enablers relate to vehicle autonomy, future vehicle connectivity and digital transformation of the automotive industry?

The outcome of this research work has put solid foundations in place for future research work. The proposed solutions can be taken to the next level in order to address the needs of the automotive industry for the next 5 to 10 years from now.

7. Conclusions

In conclusion, this research has resulted in the design and deployment of solutions for automotive embedded software development. The implementation of the solutions developed, has shown significant shift of detection of failure modes and/or software defects to earlier phases of the automotive product development process. Research findings have shown a detection of more than 50% of failure modes and/or software defects before supplier embedded software release. This is a significant improvement compared to previous programmes where failure modes and/or software defects were found after supplier software release and availability of ECU hardware. In addition to the above, evaluation results have shown substantial reduction in engineering effort and time for the creation of test cases and test scripts suitable for automated testing execution in both offline and real-time environments. Research findings demonstrated a 40% reduction in engineering effort and five to tenfold reduction in engineering time.

Data from a recent Jaguar Land Rover (JLR) vehicle programme were collected and analysed in order to realise the research challenge of early detection of failure modes and/or software defects during product development. The outcome from the analysis confirmed that compared to an internal target too many failure modes and/or software defects escape from the early stages of product development. As a result failure modes and/or software defects are found too late in the product development process resulting in expensive and time consuming engineering changes.

In the early stages of the research two studies were conducted in order to identify the importance of integration between automotive OEMs and suppliers as well as suppliers' capability for model-based development and automated testing. Research findings concluded that OEMs' capability to develop robust requirements, system

architecture designs and Electronic Control Units (ECUs) specification prior to supplier activities, and supplier undertaking to deliver robust ECUs, have significant impact on embedded software quality and the ability to detect failure modes and/or software defects at the early stages of the product development. Data collected and analysed from across the automotive supplier base, responsible for embedded software development and test, suggested that there was a lack of model-based development and automated testing during the early stages of the product development. There was overwhelming evidence corroborating the notion that the automotive supplier base was heavily relying on ECU physical prototypes being available for testing embedded software.

The research challenge of how to shift failure modes and/or software defects to the earlier phases of the automotive product development process led to the creation of a number of solutions. These solutions were driven by integrated processes, methods and tools applicable to automotive embedded software development.

A new process called Model-based Product Engineering (MBPE) was developed and deployed within JLR. Current literature shows that today “a silver bullet” for automotive embedded software development does not exist. All existing product development processes have either limitations amongst their benefits or simply are not matured yet for industrialisation. The key outcome from the literature was that the automotive industry requires pragmatic product development processes that can be understood by OEMs and their supplier base. Pragmatic product development processes have greater chances to be implemented and as a result can transform the way the automotive industry develops products. The proposed MBPE process addressed this gap, it brings together best practices from model-based development and other processes such as V-model, agile, spiral and systems engineering. The research outcome from the

development of the MBPE process was the generation of new engineering standards and design rules for JLR and its supplier base. The MBPE process has been successfully evaluated through a set of key criteria. The MBPE process has been deployed within JLR and early benefits have shown detection of failure modes and/or software defects of over 50% prior to supplier embedded software release. This is a significant shift to the early stages of the product development. In addition to that, results show a significant reduction in validation and test times driven by a combination of functional and diagnostics automated testing.

An innovative solution towards standardisation of test cases for automotive embedded software development called Design Verification Interface (DVI) was designed. Current best practice in the automotive industry drives engineers to create test scripts for specific test tools and test targets (or test platforms). The problem with this approach is that test scripts cannot be further reused for other applications which require different set of test tools and test targets. In order to overcome this problem a generic Design Verification Interface (DVI) was created in order to enable automated testing, test exchange and traceability across all Model-based Product Engineering (MBPE) process levels. The generic DVI eliminated the redundant efforts of re-writing test cases and facilitated test engineers within JLR and its supplier base with a standardised test interface.

A solution for defining test cases for all vehicle systems in a common form called Standardised Design Verification Method (SDVM) was developed. SDVM uses a semi-formal test specification approach. As a result test cases can be presented as machine readable data. The integration of DVI and SDVM drives a test solution across all MBPE process levels. The outcome of this is that test cases can be auto-generated, executed and exchanged amongst different test tools and test platforms. Evaluation results show

that test cases derived from across all key vehicle systems were successfully written in the new SDVM. Previously, these test cases would have been written in standalone and custom DVM forms.

The research activities were concluded with the development and evaluation of the proposed Platform Independent Test System (PITS). PITS is an end to end solution that supports all levels of system abstraction from the test case definition phase to the execution of automated test scripts in both offline and real time test environments. Research findings from the PITS evaluation suggested 40% engineering effort reduction for test script creation for automated testing and five to tenfold engineering time reduction compared to manual and automated testing using Commercial of the Shelf (COTS) tools. The PIT system offers sustainable solution for maintaining and archiving test scripts for future use. In addition to that the company's expenditure for test automation tools licence and maintenance is significantly reduced in the range of £1.1M over a five years period.

The MBPE process is already in production. MBPE process requirements were cascaded to all JLR departments and suppliers that are responsible for developing automotive embedded software. The DVI, SDVM and PITS are not in production yet. They need to move from their prototype phase to the development in order to be fully prepared for a release into production.

The integration of PITS with the MBPE process, the SDVM and the DVI have driven solutions for the company with the potential to significantly shift failure modes and/or software defects detection to the early stages of the product development. The outcome of this research means that JLR is in a stronger position to develop future products with better embedded software quality and reduced time to market.

8. References

Achimugu, P., Selamat, A., Ibrahim, R. and Mahrin, M. N. (2014). A systematic literature review of software requirements prioritization research. *Information and Software Technology*, 56 (6), p.568–585.

Aichernig, B. K., Lorber, F. and Tiran, S. (2012). Integrating model-based testing and analysis tools via test case exchange. In: *Theoretical Aspects of Software Engineering (TASE), 2012 Sixth International Symposium on*, 2012, IEEE, p.119–126.

Akhtar, Z. (2015). Model Based Automotive System Design: A Power Window Controller Case Study. *Model Based Automotive System Design*.

Alagar, V. S. and Periyasamy, K. (2011). *Specification of software systems*. Springer Science & Business Media.

Alegroth, E., Feldt, R. and Olsson, H. H. (2013). Transitioning manual system test suites to automated testing: An industrial case study. In: *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, 2013, IEEE, p.56–65.

Alemanni, M., Destefanis, F. and Vezzetti, E. (2011). Model-based definition design in the product lifecycle management scenario. *The International Journal of Advanced Manufacturing Technology*, 52 (1–4), p.1–14.

Altinger, H., Wotawa, F. and Schurius, M. (2014). Testing methods used in the automotive industry: results from a survey. In: *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, 2014, ACM, p.1–6.

Andrianarison, E. and Piques, J.-D. (2010). SysML for embedded automotive Systems: a practical approach. *Embedded Real Time Software and Systems ERTS*, 2010.

Apvrille, L. and Becoulet, A. (2012). Prototyping an embedded automotive system from its UML/SysML models. *Proceedings of Embedded Real Time Systems and Software (ERTSS 2012)*, p.87–124.

ASAM. (2015). *Association for Standardisation of Automation and Measuring Systems*. [Online]. Available at: http://www.asam.net/ [Accessed: 12 December 2015].

Automotive SPICE. (2015). *Automotive SPICE Version 3.0*. [Online]. Available at: http://www.automotivespice.com/ [Accessed: 12 September 2015].

AUTOSAR. (2015). *AUTomotive Open System ARchitecture*. [Online]. Available at: http://www.autosar.org/ [Accessed: 12 December 2015].

Awais, M. U., Palensky, P., Elsheikh, A., Widl, E. and Matthias, S. (2013). The high level architecture RTI as a master to the functional mock-up interface components. In: *Computing, Networking and Communications (ICNC), 2013 International Conference on*, 2013, IEEE, p.315–320.

Axelsson, J. (2001). Methods and tools for systems engineering of automotive electronic architectures. In: *DATE*, 2001, p.112.

Bak, K., Novakovic, M. and Passos, L. (2013). *Exemplar of automotive architecture with variability*.

Balaji, S. and Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2 (1), p.26–30.

Bansal, A., Muli, M. and Patil, K. (2013). Taming complexity while gaining efficiency: Requirements for the next generation of test automation tools. In: *AUTOTESTCON, 2013 IEEE*, 2013, IEEE, p.1–6.

Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. *arXiv preprint arXiv:1205.6904*.

Beckers, K., Heisel, M., Frese, T. and Hatebur, D. (2013). A structured and model-based hazard analysis and risk assessment method for automotive systems. In: *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, 2013, IEEE, p.238–247.

ter Beek, M. H., Fantechi, A., Gnesi, S. and Mazzanti, F. (2011). A state/event-based model-checking approach for the analysis of abstract system properties. *Science of Computer Programming*, 76 (2), p.119–135.

Bell, T. E. and Thayer, T. A. (1976). Software requirements: Are they really a problem? In: *Proceedings of the 2nd international conference on Software engineering*, 1976, IEEE Computer Society Press, p.61–68.

Bennett, T. L. and Wennberg, P. W. (2005). Eliminating embedded software defects prior to integration test. *Crosstalk, Journal of Defence Software Engineering*.

Bernardi, S., Flammini, F., Marrone, S., Mazzocca, N., Merseguer, J., Nardone, R. and Vittorini, V. (2013). Enabling the usage of UML in the verification of railway systems: the DAM-rail approach. *Reliability Engineering & System Safety*, 120, p.112–126.

Biggs, G., Fujiwara, K. and Anada, K. (2014). Modelling and Analysis of a Redundant Mobile Robot Architecture Using AADL. In: *Simulation, Modeling, and Programming for Autonomous Robots*, Springer, p.146–157.

Blochwitz, T., Otter, M., \AAkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D. and others. (2012). Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In: *9th International Modelica Conference*, 2012, The Modelica Association, p.173–184.

Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clau\s s, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T. and others. (2011). The functional mockup interface for tool independent exchange of simulation models. In: *Proceedings of the 8th International Modelica Conference*, 2011, Linköping University Press, p.105–114.

Boca, P., Bowen, J. P. and Siddiqi, J. I. (2010). *Formal methods: State of the art and new directions*. Springer.

Boehm, B., Lane, J. A., Koolmanojwong, S. and Turner, R. (2014). *The incremental commitment spiral model: Principles and practices for successful systems and software*. Addison-Wesley Professional.

Börjesson, E. and Feldt, R. (2012). Automated system testing using visual GUI testing tools: A comparative study in industry. In: *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, 2012, IEEE, p.350–359.

Bosch, J. and Eklund, U. (2012). Eternal embedded software: Towards innovation experiment systems. In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, 2012, Springer, p.19–31.

Boulet, P., Amyot, D. and Stepien, B. (2015). Towards the Generation of Tests in the Test Description Language from Use Case Map Models. In: *International SDL Forum*, 2015, Springer, p.193–201.

Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V. Y., Noll, T. and Roveri, M. (2010a). Safety, dependability and performance analysis of extended AADL models. *The Computer Journal*.

Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V. Y., Noll, T., Roveri, M. and Wimmer, R. (2010b). A model checker for AADL. In: *Computer Aided Verification*, 2010, Springer, p.562–565.

Braun, P., Broy, M., Houdek, F., Kirchmayr, M., Müller, M., Penzenstadler, B., Pohl, K. and Weyer, T. (2014a). Guiding requirements engineering for software-intensive embedded systems in the automotive industry. *Computer Science-Research and Development*, 29 (1), p.21–43.

Braun, P., Broy, M., Houdek, F., Kirchmayr, M., Müller, M., Penzenstadler, B., Pohl, K. and Weyer, T. (2014b). Guiding requirements engineering for software-intensive embedded systems in the automotive industry. *Computer Science-Research and Development*, 29 (1), p.21–43.

Briciu, C.-V., Filip, I. and Heininger, F. (2013). A new trend in automotive software: AUTOSAR concept. In: *Applied Computational Intelligence and Informatics (SACI), 2013 IEEE 8th International Symposium on*, 2013, IEEE, p.251–256.

Broy, M. (2006). Challenges in automotive software engineering. In: *Proceedings of the 28th international conference on Software engineering*, ICSE '06, 2006, New York, NY, USA, p.33–42.

Broy, M., Kirstan, S., Krcmar, H., Schätz, B. and Zimmermann, J. (2013). What is the benefit of a model-based design of embedded software systems in the car industry? *Software Design and Development: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*, p.310.

BSI ISO 13209. (2011). *Road vehicles - Open Test sequence eXchange format (OTX)*. BSI ISO.

BSI ISO 22901. (2011). *Road vehicles - Open Diagnostic data eXchange formay (ODX)*. BSI ISO.

Bucchiarone, A., Gnesi, S. and Pierini, P. (2005). Quality analysis of NL requirements: an industrial case study. In: *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, 2005, IEEE, p.390–394.

Buckl, C., Camek, A., Kainz, G., Simon, C., Mercep, L., Stahle, H. and Knoll, A. (2012). The software car: Building ICT architectures for future electric vehicles. In: *Electric Vehicle Conference (IEVC), 2012 IEEE International*, 2012, IEEE, p.1–8.

Cervone, H. F. (2009). Applied digital library project management: Using Pugh matrix analysis in complex decision-making situations. *OCLC Systems & Services: International digital library perspectives*, 25 (4), p.228–232.

Chakraborty, S., Lukasiewycz, M., Buckl, C., Fahmy, S., Chang, N., Park, S., Kim, Y., Leteinturier, P. and Adlkofer, H. (2012a). Embedded systems and software challenges in electric vehicles. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, 2012, p.424–429.

Chakraborty, S., Lukasiewycz, M., Buckl, C., Fahmy, S., Chang, N., Park, S., Kim, Y., Leteinturier, P. and Adlkofer, H. (2012b). Embedded systems and software challenges in electric vehicles. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, 2012, EDA Consortium, p.424–429.

Charette, R. N. (2009). This car runs on code. *IEEE Spectrum*, 46 (3), p.3.

Chaudhary, P. and Yadav, C. S. (2012). An Approach for Calculating the Effort Needed on Testing Projects. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 1 (1), p.pp–35.

Chemuturi, M. (2013). *Requirements engineering and management for software development projects*. New York, NY: Springer.

Chrissis, M. B., Konrad, M. and Shrum, S. (2003). *CMMI Guidlines for Process Integration and Product Improvement*. Addison-Wesley Longman Publishing Co., Inc.

Chung, L., Nixon, B. A., Yu, E. and Mylopoulos, J. (2012). *Non-functional requirements in software engineering*. Springer Science & Business Media.

Clark, J. O. (2009). System of systems engineering and family of systems engineering from a standards, V-model, and dual-V model perspective. In: *Systems Conference, 2009 3rd Annual IEEE*, 2009, IEEE, p.381–387.

Conrad, M., Fey, I. and Sadeghipour, S. (2005). Systematic Model-Based Testing of Embedded Automotive Software. *Electronic Notes in Theoretical Computer Science*, 111, p.13–26.

Cooper, K. M., Nasr, E. S. and Longstreet, C. S. (2014). Towards model-driven requirements engineering for serious educational games: Informal, semi-formal, and formal models. In: *Requirements Engineering: Foundation for Software Quality*, Springer, p.17–22.

Cuenot, P., Frey, P., Johansson, R., Lönn, H., Papadopoulos, Y., Reiser, M.-O., Sandberg, A., Servat, D., Kolagari, R. T., Törngren, M. and others. (2010). The EAST-

ADL Architecture Description Language for Automotive Embedded Software. In: *Model-Based Engineering of Embedded Real-Time Systems*, Springer, p.297–307.

Currie, J., Prince-Pike, A. and Wilson, D. I. (2012). Auto-code generation for fast embedded model predictive controllers. In: *Mechatronics and Machine Vision in Practice (M2VIP), 2012 19th International Conference*, 2012, IEEE, p.116–122.

Daun, M., Weyer, T. and Pohl, K. (2015). Detecting and Correcting Outdated Requirements in Function-Centered Engineering of Embedded Systems. In: *Requirements Engineering: Foundation for Software Quality*, Springer, p.65–80.

Di Guglielmo, G., Fummi, F., Pravadelli, G., Soffia, S. and Roveri, M. (2010). Semi-formal functional verification by EFSM traversing via NuSMV. In: *High Level Design Validation and Test Workshop (HLDVT), 2010 IEEE International*, 2010, IEEE, p.58–65.

Dorfman, M. and Thayer, R. H. (eds.). (2000). *Software requirements engineering*, Practitioners. Los Alamitos, California : IEEE Computer Society Press: IEEE Xplore.

Drabek, C., Pramsohler, T., Zeller, M. and Weiss, G. (2013). Interface Verification Using Executable Reference Models: An Application in the Automotive Infotainment. In: *ACESMB@ MoDELS*, 2013, Citeseer.

Duan, L., Hofer, A. and Hussmann, H. (2010). Model-based testing of infotainment systems on the basis of a graphical human-machine interface. In: *Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on*, 2010, IEEE, p.5–9.

Ebert, C. and Jones, C. (2009a). Embedded software: Facts, figures, and future. *Computer*, 42 (4), p.42–52.

Ebert, C. and Jones, C. (2009b). Embedded software: Facts, figures, and future. *Computer*, 42 (4), p.42–52.

Eckstein, J. (2013). *Agile software development with distributed teams: Staying agile in a global world*. Addison-Wesley.

Ersal, T., Fuller, H. J., Tsimhoni, O., Stein, J. L. and Fathy, H. K. (2010). Model-based analysis and classification of driver distraction under secondary tasks. *Intelligent Transportation Systems, IEEE Transactions on*, 11 (3), p.692–701.

Exel, L., Frey, G., Wolf, G. and Oppelt, M. (2014). Re-use of existing simulation models for DCS engineering via the Functional Mock-up Interface. In: *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, 2014, IEEE, p.1–4.

Fabbrini, F., Fusani, M., Gnesi, S. and Lami, G. (2001). An automatic quality evaluation for natural language requirements. In: *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ*, 1, 2001, p.4–5.

Feiler, P. H. and Gluch, D. P. (2012). *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley.

Fernandes, M. M., Rosati, A. C., Neto, D. G., Goto, F. K., Maciel, H. and Mologni, J. F. (2008). *Quality Function Deployment (QFD) and Pugh Matrix on Innovative Concept Selection: an Application in Automotive Sector*. SAE Technical Paper.

Ferreira, N., Santos, N., Machado, R. J., Fernandes, J. E. and Gasević, D. (2014). A V-model approach for business process requirements elicitation in cloud design. In: *Advanced Web Services*, Springer, p.551–578.

FMI. (2015). *Functional Mock-up Interface*. [Online]. Available at: https://www.fmi-standard.org/ [Accessed: 12 September 2015].

Föcker, F., Houdek, F., Daun, M. and Weyer, T. (2015a). *Model-based engineering of an automotive adaptive exterior lighting system: Realistic example specifications of behavioral requirements and functional design*. ICB-Research Report.

Föcker, F., Houdek, F., Daun, M. and Weyer, T. (2015b). *Model-based engineering of an automotive adaptive exterior lighting system: Realistic example specifications of behavioral requirements and functional design*. ICB-Research Report.

Fürst, S. (2010). Challenges in the design of automotive software. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, p.256–258.

Gai, P. and Violante, M. (2016). Automotive embedded software architecture in the multi-core age. In: *Test Symposium (ETS), 2016 21th IEEE European*, 2016, IEEE, p.1–8.

Gallina, B., Kashiyarandi, S., Martin, H. and Bramberger, R. (2014). Modeling a safety- and automotive-oriented process line to enable reuse and flexible process derivation. In: *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*, 2014, IEEE, p.504–509.

Gerhart, S., Craigen, D. and Ralston, T. (2012). Experience with formal methods in critical systems. *High-Integrity System Specification and Design*, p.413.

Gigante, G., Gargiulo, F. and Ficco, M. (2015). A semantic driven approach for requirements verification. In: *Intelligent Distributed Computing VIII*, Springer, p.427–436.

Gmehlich, R. and Jones, C. (2013). Experience of deployment in the automotive industry. In: *Industrial Deployment of System Engineering Methods*, Springer, p.13–26.

Goknil, A., Kurtev, I. and Van Den Berg, K. (2014). Generation and validation of traces between requirements and architecture based on formal trace semantics. *Journal of Systems and Software*, 88, p.112–137.

Góngora, H. G. C., Gaudré, T. and Tucci-Piergiovanni, S. (2013a). Towards an architectural design framework for automotive systems development. In: *Complex Systems Design & Management*, Springer, p.241–258.

Góngora, H. G. C., Gaudré, T. and Tucci-Piergiovanni, S. (2013b). Towards an architectural design framework for automotive systems development. In: *Complex Systems Design & Management*, Springer, p.241–258.

González-Huerta, J., Insfran, E., Abrahão, S. and McGregor, J. D. (2012). Non-functional requirements in model-driven software product line engineering. In: *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, 2012, ACM, p.6.

Gorschek, T., Gomes, A., Pettersson, A. and Torkar, R. (2012). Introduction of a process maturity model for market-driven product management and requirements engineering. *Journal of software: Evolution and Process*, 24 (1), p.83–113.

Goto, M. (2013). Innovation of automotive software development. In: *Proceedings of the 17th International Software Product Line Conference*, 2013, ACM, p.5–6.

Graaf, B., Lormans, M. and Toetenel, H. (2003). Embedded software engineering: the state of the practice. *IEEE software*, 20 (6), p.61–69.

Grady, R. B. (1992). *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc.

Grimm, K. (2003). Software technology in an automotive company: major challenges. In: *Proceedings of the 25th international conference on Software Engineering*, 2003, p.498–503.

Grönninger, H., Krahn, H., Pinkernell, C. and Rumpe, B. (2014). Modeling variants of automotive systems using views. *arXiv preprint arXiv:1409.6629*.

Grossmann, J., Fey, I., Krupp, A., Conrad, M., Wewetzer, C. and Mueller, W. (2006). Testml-a test exchange language for model-based testing of embedded software. In: *Automotive Software Workshop*, 2006, Springer, p.98–117.

Grossmann, J. and Müller, W. (2006). A formal behavioral semantics for TestML. In: *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on*, 2006, IEEE, p.441–448.

Grosu, R., Klein, C., Rumpe, B. and Broy, M. (1996). *State transition diagrams*.

Hanna, M., El-Haggar, N. and Sami, M. (2013). Reducing Testing Effort using Automation. *International Journal of Computer Applications*, 81 (8).

Hanselmann, H. (2008). Challenges in automotive software engineering. In: *Companion of the 30th international conference on Software engineering*, 2008, p.888–888.

Harris, A., Motato, E., Mohammadpour, M., Theodossiades, S., Rahnejat, H., Kelly, P., O'Mahony, M. and Struve, B. (2016). *Concept selection for clutch nonlinear absorber using PUGH matrix*.

He, N., Rümmer, P. and Kroening, D. (2011). Test-case generation for embedded simulink via formal concept analysis. In: *Proceedings of the 48th Design Automation Conference*, 2011, ACM, p.224–229.

Henzinger, T. A. and Sifakis, J. (2006). The embedded systems design challenge. In: *FM 2006: Formal Methods*, Springer, p.1–15.

Hess, S., Gross, A., Maier, A., Orfgen, M. and Meixner, G. (2012). Standardizing model-based in-vehicle infotainment development in the German automotive industry. In: *Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 2012, ACM, p.59–66.

Hinchey, M. G. and Bowen, J. P. (2012). *Industrial-strength formal methods in practice*. Springer Science & Business Media.

Holt, J., Perry, S. A. and Brownsword, M. (2011). *Model-Based Requirements Engineering*, Computing. Stevenage: IET.

Holtmann, J., Meyer, J. and Meyer, M. (2011). A Seamless Model-Based Development Process for Automotive Systems. In: *Software Engineering (Workshops)*, 2011, p.79–88.

Hoyos Velasco, F. E., Garcia, N. T. and Garcia, F. A. (2012). Rapid Control Prototyping of a permanent magnet DC motor using non-linear sliding control ZAD and FPIC. In: *Circuits and Systems (LASCAS), 2012 IEEE Third Latin American Symposium on*, 2012, IEEE, p.1–4.

Hu, M., Huang, Y., Zhao, C., Di, X., Liu, B. and Li, H. (2014). Model-based development and automatic code generation of powertrain control system. In: *Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), 2014 IEEE Conference and Expo*, 2014, IEEE, p.1–4.

Huang, Y., McMurran, R., Amor-Segan, M., Dhadyalla, G., Jones, R. P., Bennett, P., Mouzakitis, A. and Kieloch, J. (2010a). Development of an automated testing system for vehicle infotainment system. *The International Journal of Advanced Manufacturing Technology*, 51 (1–4), p.233–246.

Huang, Y., McMurran, R., Amor-Segan, M., Dhadyalla, G., Jones, R. P., Bennett, P., Mouzakitis, A. and Kieloch, J. (2010b). Development of an automated testing system for vehicle infotainment system. *The International Journal of Advanced Manufacturing Technology*, 51 (1–4), p.233–246.

Hull, E., Jackson, K. and Dick, J. (2010). *Requirements engineering*. Springer Science & Business Media.

Hull, E., Jackson, K. and Dick, J. (2011). *Requirements engineering*. 3rd ed. London: Springer.

Inagaki, H. (2013). *Development of the Clutch Controller for the Hybrid System using Automatic Code Generation*. SAE Technical Paper.

Insfran, E., Chastek, G., Donohoe, P. and do Prado Leite, J. C. S. (2014). Requirements engineering in software product line engineering. *Requirements Engineering*, 19 (4), p.331–332.

Iqbal, M. Z., Arcuri, A. and Briand, L. (2015). Environment modeling and simulation for automated testing of soft real-time embedded software. *Software & Systems Modeling*, 14 (1), p.483–524.

ISO 26262. (2011). *ISO 26262 Road Vehicles - Functional Safety*. [Online]. Available at: http://www.iso.org/iso/home.html [Accessed: 12 September 2015].

Jaguar Land Rover. (2014). *Jaguar Land Rover corporate website*. [Online]. Available at: www.jaguarlandrover.com [Accessed: 13 January 2014].

Javed, A. Z., Strooper, P. A. and Watson, G. N. (2007). Automated generation of test cases using model-driven architecture. In: *Automation of Software Test, 2007. AST'07. Second International Workshop on*, 2007, IEEE, p.3–3.

JLR-APC. (2012). *Electrical Architecture Diagram - Range Rover Evoque*. Jaguar Land Rover.

Junior, E. O., Gimenes, I. M. and Maldonado, J. C. (2010). Systematic management of variability in UML-based software product lines. *Journal of Universal Computer Science*, 16 (17), p.2374–2393.

Kakade, R., Murugesan, M., Perugu, B. and Nair, M. (2010). Model-Based development of automotive electronic climate control software. In: *Modelling Foundations and Applications*, Springer, p.144–155.

Karsai, G. (2006). Automotive software: A challenge and opportunity for model-based software development. In: *Automotive Software–Connected Services in Mobile Networks*, Springer, p.103–115.

Kaur, H. and Singh, P. (2011). UML (Unified Modeling Language): standard language for software architecture development. In: *International Symposium on Computing, Communication, and Control*, 2011.

Keller, G., Scheer, A.-W. and Nüttgens, M. (1992). *Semantische Proze\s smodellierung auf der Grundlage' Ereignisgesteuerter Proze\s sketten (EPK)'*. Inst. für Wirtschaftsinformatik.

Keranen, J. S. and Räty, T. D. (2012). Model-based testing of embedded systems in hardware in the loop environment. *Software, IET*, 6 (4), p.364–376.

Khabbaz Saberi, A., Luo, Y., Cichosz, F. P., van den Brand, M. and Jansen, S. (2015). An approach for functional safety improvement of an existing automotive system. In: *Systems Conference (SysCon), 2015 9th Annual IEEE International*, 2015, IEEE, p.277–282.

Kindler, E. (2006). On the semantics of EPCs: Resolving the vicious circle. *Data & Knowledge Engineering*, 56 (1), p.23–40.

Kossiakoff, A., Sweet, W. N., Seymour, S. and Biemer, S. M. (2011). *Systems engineering principles and practice*. John Wiley & Sons.

Krammer, M., Martin, H., Karner, M., Watzenig, D. and Fuchs, A. (2013). *System Modeling for Integration and Test of Safety-Critical Automotive Embedded Systems*. SAE Technical Paper.

Krieg, A., Preschern, C., Grinschgl, J., Steger, C., Kreiner, C., Weiss, R., Bock, H. and Haid, J. (2013). Power And Fault Emulation for Software Verification and System

Stability Testing in Safety Critical Environments. *Industrial Informatics, IEEE Transactions on*, 9 (2), p.1199–1206.

Krogstie, J. (2012). *Model-based development and evolution of information systems: A Quality Approach*. Springer.

Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

Kumar, D. and Mishra, K. K. (2016). The Impacts of Test Automation on Software's Cost, Quality and Time to Market. *Procedia Computer Science*, 79, p.8–15.

Lami, G. and Falcini, F. (2014). Automotive SPICE Assessments in Safety-Critical Contexts: An Experience Report. In: *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, 2014, IEEE, p.497–502.

Lami, G., Gnesi, S., Fabbrini, F., Fusani, M. and Trentanni, G. (2004). An automatic tool for the analysis of natural language requirements. *Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre*.

Lano, K. and Haughton, H. (1996). *Specification in B: An introduction using the B toolkit*. World Scientific.

Laplante, P. A. (2013a). *Requirements engineering for software and systems*. CRC Press.

Laplante, P. A. (2013b). *Requirements engineering for software and systems*. CRC Press.

Larsen, P. G., Plat, N. and Toetenel, H. (1994). A formal semantics of data flow diagrams. *Formal aspects of Computing*, 6 (6), p.586–606.

Lazic, L. and Mastorakis, N. (2008). Cost effective software test metrics. *WSEAS Transactions on Computers*, 7 (6), p.599–619.

Lee, E. A. (2000). What's ahead for embedded software? *Computer*, 33 (9), p.18–26.

Lee, K. J., Ki, Y. H., Cheon, J. S., Hwang, G. and Ahn, H. S. (2014). Approach to functional safety-compliant ECU design for electro-mechanical brake systems. *International journal of automotive technology*, 15 (2), p.325–332.

Lee, K., Park, I., SunWoo, M. and Lee, W. (2013). AUTOSAR-ready light software architecture for automotive embedded control systems. *Transactions of the Korean Society of Automotive Engineers*, 21 (1), p.68–77.

Levendovszky, T., Balasubramanian, D., Narayanan, A., Shi, F., van Buskirk, C. and Karsai, G. (2014). A semi-formal description of migrating domain-specific models with evolving domains. *Software & Systems Modeling*, 13 (2), p.807–823.

Li, L. and Tang, S. (2013). Development of Controller Diagnostic System Based on ODX. In: *Proceedings of the FISITA 2012 World Automotive Congress*, 2013, Springer, p.341–349.

Li, Y.-T. S. and Malik, S. (2012). *Performance analysis of real-time embedded software*. Springer Science & Business Media.

Liebel, G., Marko, N., Tichy, M., Leitner, A. and Hansson, J. (2014). Assessing the state-of-practice of model-based engineering in the embedded systems domain. In: *Model-Driven Engineering Languages and Systems*, Springer, p.166–182.

Liebezeit, J. and Serway, B. (2012). Software-in-the-Loop using virtual CAN busses. Current solutions and challenges. In: *5th Simulation and Testing symposium*, 2012, p.10–11.

Liggesmeyer, P. and Trapp, M. (2009a). Trends in embedded software engineering. *Software, IEEE*, 26 (3), p.19–25.

Liggesmeyer, P. and Trapp, M. (2009b). Trends in embedded software engineering. *IEEE software*, 26 (3), p.19–25.

Lightsey, B. (2001). *Systems engineering fundamentals*. DTIC Document.

Lind, K. and Heldal, R. (2012). A practical approach to size estimation of embedded software components. *IEEE Transactions on Software Engineering*, 38 (5), p.993–1007.

Liu, S., Tamai, T. and Nakajima, S. (2011). A framework for integrating formal specification, review, and testing to enhance software reliability. *International Journal of Software Engineering and Knowledge Engineering*, 21 (2), p.259–288.

Liu, Y., Li, Y. Q. and Zhuang, R. K. (2013a). The Application of Automatic Code Generation Technology in the Development of the Automotive Electronics Software. In: *Applied Mechanics and Materials*, 321, 2013, Trans Tech Publ, p.1574–1577.

Liu, Y., Li, Y. Q. and Zhuang, R. K. (2013b). The Application of Automatic Code Generation Technology in the Development of the Automotive Electronics Software. In: *Applied Mechanics and Materials*, 321, 2013, Trans Tech Publ, p.1574–1577.

Losada, B., Urretavizcaya, M. and Fernández-Castro, I. (2013). A guide to agile development of interactive software with a 'User Objectives'-driven methodology. *Science of Computer Programming*, 78 (11), p.2268–2281.

Lutz, R. R. (1993). Analyzing software requirements errors in safety-critical, embedded systems. In: *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, 1993, IEEE, p.126–133.

Ma, Y., Yu, H., Gautier, T., Le Guernic, P., Talpin, J.-P., Besnard, L. and Heitz, M. (2013). Toward polychronous analysis and validation for timed software architectures in aadl. In: *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013, EDA Consortium, p.1173–1178.

Mader, R., Grie\s snig, G., Armengaud, E., Leitner, A., Kreiner, C., Bourrouilh, Q., Steger, C. and Wei\s s, R. (2012). A bridge from system to software development for safety-critical automotive embedded systems. In: *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, 2012, IEEE, p.75–79.

Mahmoud, S. S. and Ahmad, I. (2013). A green model for sustainable software engineering. *International Journal of Software Engineering and Its Applications*, 7 (4), p.55–74.

Mallamo, F., Millo, F. and Rolando, L. (2014). Model-based development and calibration of last generation diesel powertrains for passenger cars. *International Journal of Powertrains*, 3 (1), p.52–74.

Marinescu, R. (2014). *Model-checking and Model-based Testing of Automotive Embedded Systems: Starting from the System Architecture*.

Marinescu, R., Saadatmand, M., Bucaioni, A., Seceleanu, C. and Pettersson, P. (2014). A model-based testing framework for automotive embedded systems. In: *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, 2014, IEEE, p.38–47.

Marko, N., Liebel, G., Sauter, D., Lodwich, A., Tichy, M., Leitner, A. and Hansson, J. (2014). *Model-Based Engineering for Embedded Systems in Practice*.

MARTE. (2015). *OMG. MARTE: Modeling and Analysis of Real-Time and Embedded Systems*. [Online]. Available at: http://www.omg.org/spec/MARTE/1.1/ [Accessed: 9 December 2015].

Mathur, S. and Malik, S. (2010). Advancements in the V-Model. *International Journal of Computer Applications*, 1 (12).

Matinnejad, R., Nejati, S., Briand, L., Bruckmann, T. and Poull, C. (2013). Automated Model-in-the-Loop Testing of Continuous Controllers Using Search. In: *Search Based Software Engineering*, Springer, p.141–157.

Maurer, M. and Winner, H. (2013a). *Automotive Systems Engineering*. Springer.

Maurer, M. and Winner, H. (2013b). *Automotive systems engineering*. Springer.

McUmber, W. E. and Cheng, B. H. (2001). A general framework for formalizing UML with formal languages. In: *Proceedings of the 23rd international conference on Software engineering*, 2001, IEEE Computer Society, p.433–442.

Mellegard, N., Staron, M. and Torner, F. (2012). A light-weight defect classification scheme for embedded automotive software and its initial evaluation. In: *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on*, 2012, IEEE, p.261–270.

Mendling, J., Neumann, G. and Nüttgens, M. (2005). Yet another event-driven process chain. In: *Business Process Management*, Springer, p.428–433.

Messnarz, R., Sokic, I., Habel, S., König, F. and Bachmann, O. (2011). Extending Automotive SPICE to Cover Functional Safety Requirements and a Safety Architecture. In: *Systems, Software and Service Process Improvement*, Springer, p.298–307.

Mjeda, A. and Hinchey, M. (2015). Requirement-centric reactive testing for safety-related automotive software. In: *Requirements Engineering and Testing (RET), 2015 IEEE/ACM 2nd International Workshop on*, 2015, IEEE, p.5–8.

Mohalik, S., Gadkari, A. A., Yeolekar, A., Shashidhar, K. C. and Ramesh, S. (2014). Automatic test case generation from simulink/stateflow models using model checking. *Software Testing, Verification and Reliability*, 24 (2), p.155–180.

Molloy, M. K. (1982). Performance analysis using stochastic Petri nets. *Computers, IEEE Transactions on*, 100 (9), p.913–917.

Mondragon, C. C., Mondragon, A. C., Miller, R. and Mondragon, E. C. (2009). Managing technology for highly complex critical modular systems: The case of automotive by-wire systems. *International Journal of Production Economics*, 118 (2), p.473–485.

Montazeri-Gh, M., Nasiri, M. and Jafari, S. (2011). Real-time multi-rate HIL simulation platform for evaluation of a jet engine fuel controller. *Simulation Modelling Practice and Theory*, 19 (3), p.996–1006.

Mossinger, J. (2010). Software in automotive systems. *Software, IEEE*, 27 (2), p.92–94.

Mubeen, S., Maki-Turja, J. and Sjodin, M. (2014). Towards Extraction of Interoperable Timing Models from Component-Based Vehicular Distributed Embedded Systems. In: *Information Technology: New Generations (ITNG), 2014 11th International Conference on*, 2014, IEEE, p.655–659.

Munassar, N. M. A. and Govardhan, A. (2010). A comparison between five models of software engineering. *IJCSI*, 5, p.95–101.

Murphy, B., Wakefield, A. and Friedman, J. (2008). Best practices for verification, validation, and test in model-based design. *The Mathworks, Inc*.

Mussbacher, G., Amyot, D. and Whittle, J. (2013). Composing goal and scenario models with the aspect-oriented user requirements notation based on syntax and semantics. In: *Aspect-Oriented Requirements Engineering*, Springer, p.77–99.

Natterer, D., Ströbele, T. and Krauss, F. (2014). ODX process from the perspective of an automotive supplier. In: *14. Internationales Stuttgarter Symposium*, 2014, Springer, p.727–737.

Navet, N. and Simonot-Lion, F. (2008). *Automotive embedded systems handbook*. CRC press.

Olivé, A. (2007). State transition diagrams. *Conceptual Modeling of Information Systems*, p.299–323.

OMG. (2015). *SysML*. [Online]. Available at: http://www.omgsysml.org/ [Accessed: 12 September 2015].

Oshana, R. and Kraeling, M. (2013). *Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications*. Newnes.

Osswald, S., Sheth, P. and Tscheligi, M. (2013). Hardware-in-the-loop-based evaluation platform for automotive instrument cluster development (EPIC). In:

Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems, 2013, ACM, p.323–332.

Pahl, G. and Beitz, W. (2013). *Engineering design: a systematic approach*. Springer Science & Business Media.

Papajorgji, P. J. and Pardalos, P. M. (2014). *Software engineering techniques applied to agricultural systems: an object-oriented and UML approach*. Springer.

Parreiras, F. S. and Staab, S. (2010). Using ontologies with UML class-based modeling: The TwoUse approach. *Data & Knowledge Engineering*, 69 (11), p.1194–1207.

Patel, U. A. and Jain, N. K. (2013). New Idea In Waterfall Model For Real Time Software Development. In: *International Journal of Engineering Research and Technology*, 2, 2013, ESRSA Publications.

Peters, H., Knieke, C., Brox, O., Jauns-Seyfried, S., Krämer, M. and Schulze, A. (2014). A Test-Driven Approach for Model-Based Development of Powertrain Functions. In: *Agile Processes in Software Engineering and Extreme Programming*, Springer, p.294–301.

Petersen, K., Wohlin, C. and Baca, D. (2009). The waterfall model in large-scale development. In: *Product-focused software process improvement*, Springer, p.386–400.

Petre, M. (2013). UML in practice. In: *Proceedings of the 2013 International Conference on Software Engineering*, 2013, IEEE Press, p.722–731.

Petrenko, A., Timo, O. N. and Ramesh, S. (2015a). Model-based testing of automotive software: Some challenges and solutions. In: *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, 2015, IEEE, p.1–6.

Petrenko, A., Timo, O. N. and Ramesh, S. (2015b). Model-based testing of automotive software: Some challenges and solutions. In: *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, 2015, IEEE, p.1–6.

Petry, E. (2010). How to Upgrade SPICE-Compliant Processes for Functional Safety. In: *Tutorial, SPICE Conference*, 2010.

Pohl, K. (2010). *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated.

Polzer, A., Merschen, D., Botterweck, G., Pleuss, A., Thomas, J., Hedenetz, B. and Kowalewski, S. (2012a). Managing complexity and variability of a model-based embedded software product line. *Innovations in Systems and Software Engineering*, 8 (1), p.35–49.

Polzer, A., Merschen, D., Botterweck, G., Pleuss, A., Thomas, J., Hedenetz, B. and Kowalewski, S. (2012b). Managing complexity and variability of a model-based embedded software product line. *Innovations in Systems and Software Engineering*, 8 (1), p.35–49.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Pretschner, A., Broy, M., Kruger, I. H. and Stauner, T. (2007). Software engineering for automotive systems: A roadmap. In: *Future of Software Engineering, 2007. FOSE'07*, 2007, p.55–71.

Pugh, S. (1991). *Total design: integrated methods for successful product engineering*. Addison-Wesley Wokingham.

Pugh, S. (2009). *The Systems Engineering Tool Box*.

Qian, K., Den Haring, D. and Cao, L. (2009). *Embedded software development with C*. Springer.

Quirk, W. J. (2012). *Verification and validation of real-time software*. Springer Science & Business Media.

Qureshi, T. N., Chen, D.-J., Persson, M. and Törngren, M. (2014). On integrating EAST-ADL and UPPAAL for embedded system architecture verification. In: *Embedded Systems Development*, Springer, p.85–99.

Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M. and Törner, F. (2013). Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model based Development. In: *ICSOFT*, 2013, p.251–257.

Rashid, U. (2014). A Methodological Approach: Formal Specification Of Quality Attributes Modeling Approcahes In The Waterfall Process Model. *VAWKUM Transaction on Computer Sciences*, 3 (2), p.1–6.

Reinhardt, D., Kaule, D. and Kucera, M. (2013). *Achieving a scalable e/e-architecture using autosar and virtualization*. SAE Technical Paper.

Riid, A., Preden, J., Pahtma, R., Serg, R. and Lints, T. (2015). Automatic Code Generation for Embedded Systems from High-Level Models. *Elektronika ir Elektrotechnika*, 95 (7), p.33–36.

Ross, D. T. (1977). Structured analysis (SA): A language for communicating ideas. *Software Engineering, IEEE Transactions on*, (1), p.16–34.

Royce, W. W. (1970). Managing the development of large software systems. In: *proceedings of IEEE WESCON*, 26, 1970, Los Angeles, p.328–388.

Saleh, K. and Al-Zarouni, A. (2004). Capturing non-functional software requirements using the user requirements notation. In: *The 2004 International Research Conference on Innovations in Information Technology*, 2004, p.222–230.

Sangiovanni-Vincentelli, A. (2000). Automotive electronics: Trends and challenges. In: *SAE Conference Proceedings*, 2000, p.295–308.

Schick, B. and Paulweber, M. (2015). Model-based development methods–What can chassis and powertrain development learn from each other? In: *6th International Munich Chassis Symposium 2015*, 2015, Springer, p.35–36.

Schieferdecker, I. (2012). Model-based testing. *IEEE software*, 29 (1), p.14.

Schmidt, K., Tröger, P., Kroll, H.-M., Bünger, T., Krueger, F. and Neuhaus, C. (2014). Adapted Development Process for Security in Networked Automotive Systems. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, 7 (2014-01–0334), p.516–526.

Schnabler, M. and Stifter, C. (2014). *Model-Based Design Methods for the Development of Transmission Control Systems*. SAE Technical Paper.

Schulze, M., Weiland, J. and Beuche, D. (2012). Automotive model-driven development and the challenge of variability. In: *Proceedings of the 16th International Software Product Line Conference-Volume 1*, 2012, ACM, p.207–214.

Selic, B. and Gérard, S. (2013). *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*. Elsevier.

Semmens, L. T., France, R. B. and Docker, T. W. G. (1992). Integrated structured analysis and formal specification techniques. *The Computer Journal*, 35 (6), p.600–610.

September, A. (1990). IEEE standard glossary of software engineering terminology. *Office*, 121990 (1), p.1.

Shahbakhti, M., Amini, M. R., Li, J., Asami, S. and Hedrick, J. K. (2015). Early Model-Based Design and Verification of Automotive Control System Software Implementations. *Journal of Dynamic Systems, Measurement, and Control*, 137 (2), p.21006.

Shahbakhti, M., Li, J. and Hedrick, J. K. (2012). Early model-based verification of automotive control system implementation. In: *American Control Conference (ACC), 2012*, 2012, IEEE, p.3587–3592.

Shahrokni, A. and Feldt, R. (2013). A systematic review of software robustness. *Information and Software Technology*, 55 (1), p.1–17.

Sharma, A. (2005). Collaborative product innovation: integrating elements of CPI via PLM framework. *Computer-Aided Design*, 37 (13), p.1425–1434.

Sharma, R. M. (2014). Quantitative Analysis of Automation and Manual Testing. *development*, 4 (1).

Shoval, P. (1988). ADISSA: Architectural design of information systems based on structured analysis. *Information systems*, 13 (2), p.193–210.

Shukla, A. and PVK, C. R. (2015). Model Based Design of Hybrid Electric Vehicle (HEV) Powertrain. *Journal of Advanced Research in Electrical Engineering and Technology*, 1 (2), p.5–10.

Siegl, S., Hielscher, K.-S., German, R. and Berger, C. (2011). Formal specification and systematic model-driven testing of embedded automotive systems. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, 2011, IEEE, p.1–6.

Sikandar-gani, S. B. (2003). User Requirement Notation (URN). *Graduate Student, Department of Electrical and Computer Engineering, Mississippi State University, MS, USA*.

Song, I.-Y., Evans, M. and Park, E. K. (1995). A comparative analysis of entity-relationship diagrams. *Journal of Computer and Software Engineering*, 3 (4), p.427–459.

Sporer, H. (2015). A model-based domain-specific language approach for the automotive E/E-System design. In: *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, 2015, ACM, p.357–362.

Stauder, S., Plöger, M. and Müller, I. S. (2014). Model-Based Controller and Function Development for Mechatronic Steering Systems. *Auto Tech Review*, 3 (7), p.30–35.

Steiner, M., Blaschke, M., Philipp, M. and Schweigert, T. (2012). Make test process assessment similar to software process assessment—the Test SPICE approach. *Journal of Software: Evolution and Process*, 24 (5), p.471–480.

Studnia, I., Nicomette, V., Alata, E., Deswarte, Y., Kaâniche, M. and Laarouchi, Y. (2013). Survey on security threats and protection mechanisms in embedded automotive networks. In: *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, 2013, IEEE, p.1–12.

Subke, P. (2014). *Internationally Standardized Technology for the Diagnostic Communication of External Test Equipment with Vehicle ECUs*. SAE Technical Paper.

Subke, P. and Eberl, M. (2015). *SAE meets ISO: Description of SAE J1939-73 on SAE J1939-21 in ODX 2.2. 0 Format (ISO 22901)*. SAE Technical Paper.

Tang, D. and Qian, X. (2008). Product lifecycle management for automotive development focusing on supplier integration. *Computers in Industry*, 59 (2–3), p.288–295.

Thakker, A., Jarvis, J., Buggy, M. and Sahed, A. (2009). 3DCAD conceptual design of the next-generation impulse turbine using the Pugh decision-matrix. *Materials & Design*, 30 (7), p.2676–2684.

Thorén, L. and Burgren, M. (2015). *Comparing the Outcomes of Two Decision Support Models: The Analytical Hierarchy Process and Pugh Matrix Analysis: Using an actual multi-criteria decision-making situation*.

Thun, J.-H. and Hoenig, D. (2011). An empirical analysis of supply chain risk management in the German automotive industry. *International Journal of Production Economics*, 131 (1), p.242–249.

Tran, H. N., Singhoff, F., Rubini, S. and Boukhobza, J. (2014). Instruction cache in hard real-time systems: modeling and integration in scheduling analysis tools with AADL. In:

Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on, 2014, IEEE, p.104–111.

Umar, M. and Khan, M. N. (2012). A Framework to Separate Non-Functional Requirements for System Maintainability. *Kuwait Journal of Science & Engineering*, 39 (1), p.211–231.

Unger, D. and Eppinger, S. (2011). Improving product development process design: a method for managing information flows, risks, and iterations. *Journal of Engineering Design*, 22 (10), p.689–699.

Völter, M., Stahl, T., Bettin, J., Haase, A. and Helsen, S. (2013). *Model-driven software development: technology, engineering, management*. John Wiley & Sons.

Vora, A., Wu, H., Wang, C., Qian, Y., Shaver, G., Motevalli, V., Meckl, P., Wasynczuk, O. and Zhang, H. (2014). *Development of a SIL, HIL and Vehicle Test-Bench for Model-Based Design and Validation of Hybrid Powertrain Control Strategies*. SAE Technical Paper.

Walter, S., Rettberg, A. and Kreutz, M. (2015). Towards formalized model-based requirements for a seamless design approach in safety-critical systems development. In: *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2015 IEEE International Symposium on*, 2015, IEEE, p.111–115.

Walters, J., Harianto, C., Kelly, E. and Sugiarto, T. (2014). Application of Auto-Coding for Rapid and Efficient Motor Control Development. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, 7 (2014-01–0305), p.481–490.

Wang, C. J., Ge, L. and Lee, T. Y. (2014). Automotive ECU Software Design Based on AUTOSAR. In: *Applied Mechanics and Materials*, 577, 2014, Trans Tech Publ, p.1034–1037.

Wang, C., Pastore, F., Goknil, A., Briand, L. and Iqbal, Z. (2015). Automatic generation of system test cases from use case specifications. In: *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, ACM, p.385–396.

Wang, J., Song, C. and Jin, L. (2010). Modeling and simulation of automotive four-channel hydraulic ABS based on AMESim and Simulink/Stateflow. In: *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on*, 2010, IEEE, p.1–4.

Waugh, K., Thomas, P. and Smith, N. (2004). *Toward the automated assessment of entity-relationship diagrams*.

Weber, J. (2014). *Automotive development processes*. Springer.

Wei, J., Mouzakitis, A., Wang, J. and Sun, H. (2011). Vehicle windscreen wiper mathematical model development and optimisation for model based hardware-in-the-loop simulation and control. In: *Automation and Computing (ICAC), 2011 17th International Conference on*, 2011, IEEE, p.207–212.

Weibel, M., Schmei\s ser, V. and Hofmann, F. (2014). Model-Based Approaches to Exhaust Aftertreatment System Development. In: *Urea-SCR Technology for deNOx After Treatment of Diesel Exhausts*, Springer, p.691–707.

Weilkiens, T. (2011a). *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann.

Weilkiens, T. (2011b). *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann.

Wiegers, K. and Beatty, J. (2013a). *Software requirements*. Pearson Education.

Wiegers, K. and Beatty, J. (2013b). *Software requirements*. Pearson Education.

Wiengarten, F., Humphreys, P., Cao, G., Fynes, B. and McKittrick, A. (2010). Collaborative supply chain practices and performance: exploring the key role of information quality. *Supply Chain Management: An International Journal*, 15 (6), p.463–473.

Woodcock, J. and Davies, J. (1996). *Using Z. Specification, refinement, and proof*.

Wu, H. (2014). *Model-based powertrain design and control system development for the ideal all-wheel drive electric vehicle*.

Yeaton, M. (2004). *Managing the challenges of automotive embedded software development using model-based methods for design and specification*. Warrendale, PA: SAE International.

Yu, H., Joshi, P., Talpin, J.-P., Shukla, S. and Shiraishi, S. (2015). *The Challenge of Interoperability: Model-Based Integration for Automotive Control Software*.

Yu, H., Talpin, J.-P., Shukla, S., Joshi, P. and Shiraishi, S. (2014). Towards an Architecture-Centric Approach dedicated to Model-Based Virtual Integration for Embedded Software Systems. In: *ACVI 2014–Architecture Centric Virtual Integration Workshop Proceedings*, 2014, p.79.

Yun, H. and Lee, S. (2011). ODX-based vehicle mobile gateway for EV telematics. In: *ICT Convergence (ICTC), 2011 International Conference on*, 2011, IEEE, p.672–675.

Zafar, N. A. and Alhumaidan, F. (2011). Transformation of class diagrams into formal specification. *International Journal Computer Science and Network Security*, 11 (5), p.289–295.

Zander, J., Schieferdecker, I. and Mosterman, P. J. (2011). *Model-based testing for embedded systems*. New York, NY, USA: CRC Press.

Zhuang, W., Dai, D. and Yu, R. J. (2014). Design and Implementation of Embedded Software for Context-Aware Wearable Body Sensor Networks. In: *Applied Mechanics and Materials*, 556, 2014, Trans Tech Publ, p.5497–5500.

Appendix A. Event-driven process chain

Table 27 shows the Event-driven Process (EPC) chain symbols used to depict the engineering activities of the model-based product engineering process.

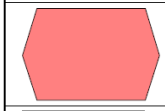
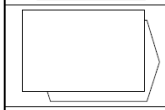
Symbol	Name	Operation
	Function	Function is an active element, which describes an activity, task or procedure. A change in the process is represented by function.
	Event	Event is a passive element, which represents a state or a status in the process.
	Process Path	Link to predecessor or successor of process.
	XOR (Exclusive OR) Connector	Only one of the paths can be followed.
	OR Connector	At least one of the paths can be followed.
	AND Connector	All paths must be followed.

Table 27. EPC symbols.

Appendix B. Model-based testing concepts

B.1. Model-in-the-loop concept

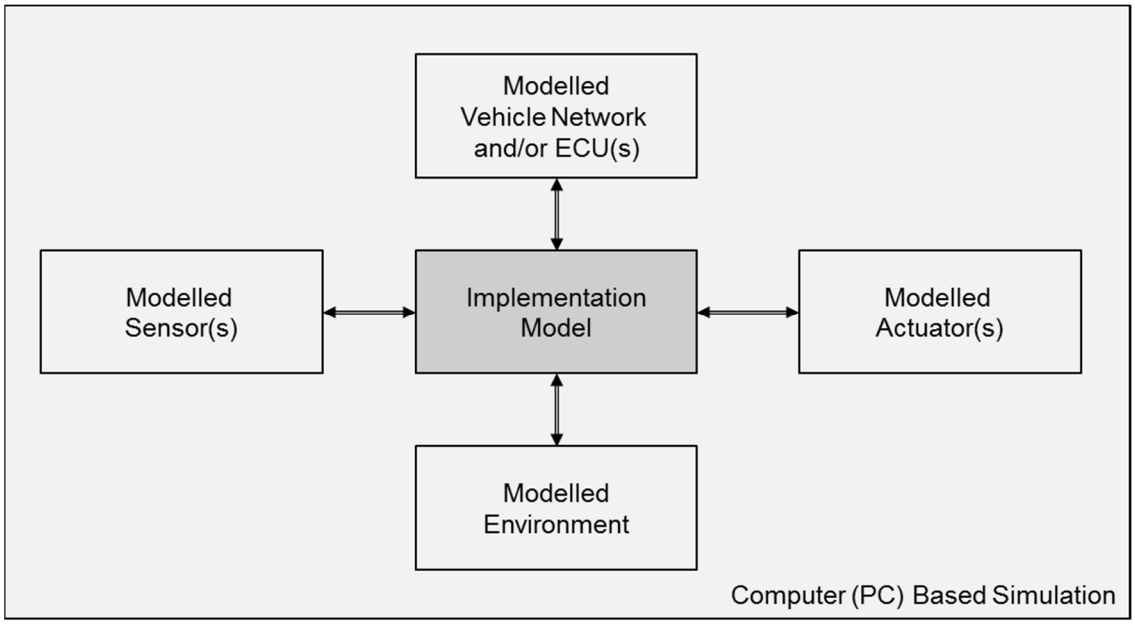


Figure 40. Model-in-the-Loop (MIL) concept.

B.2. Software-in-the-loop concept

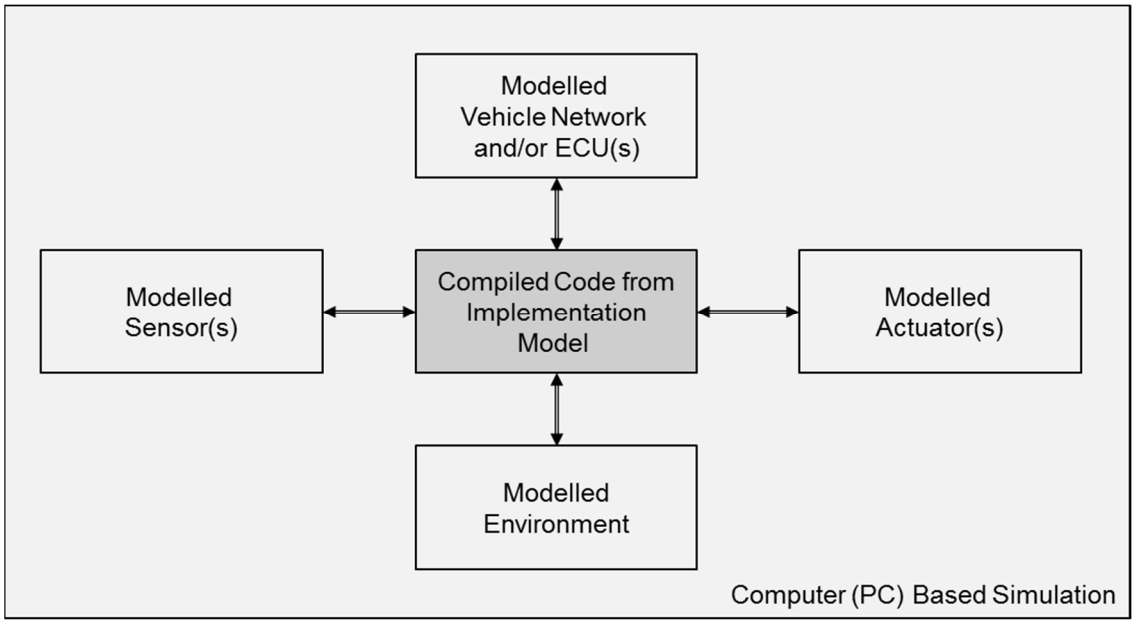


Figure 41. Software-in-the-Loop (SIL) concept.

B.3. Rapid controller prototyping concept

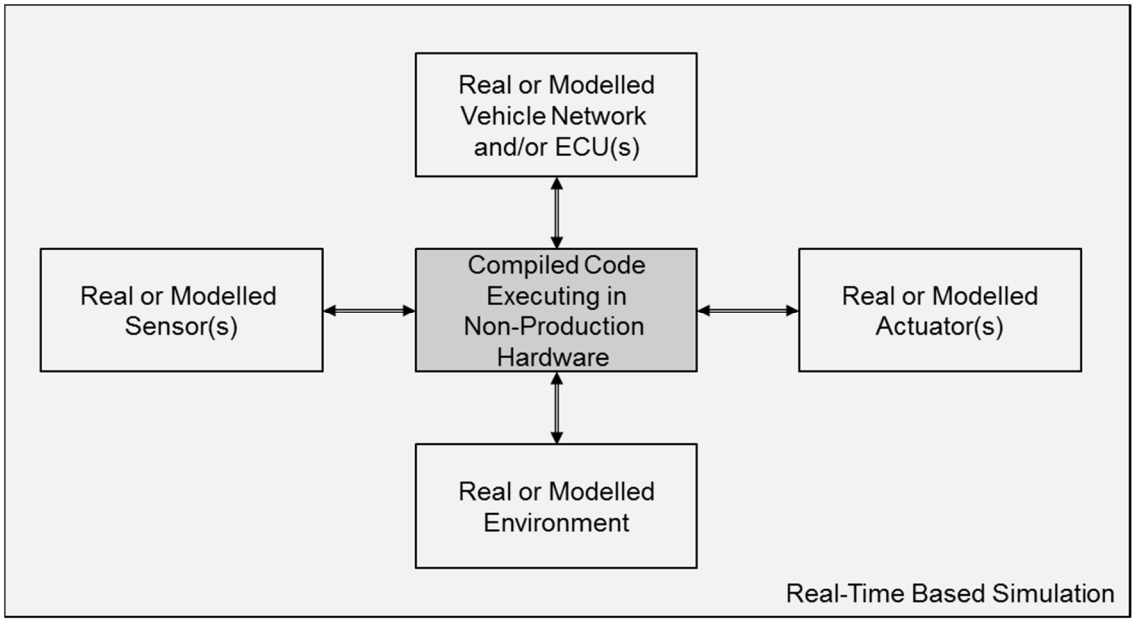


Figure 42. Rapid Controller Prototyping (RCP) concept.

B.4. Hardware-in-the-loop concept

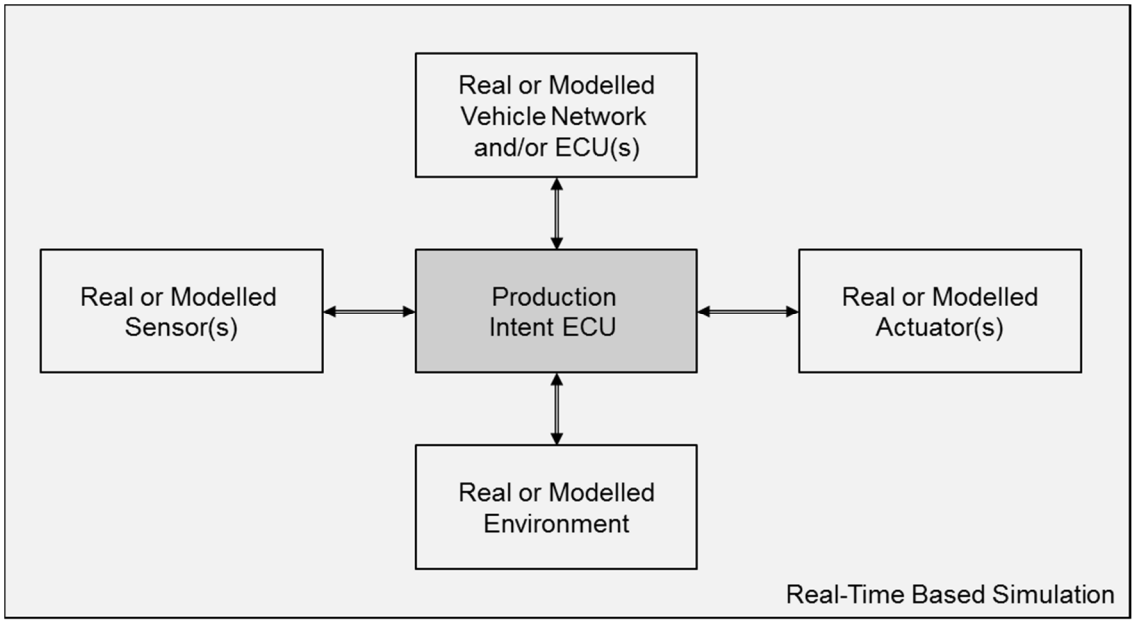


Figure 43. Hardware-in-the-Loop (HIL) concept.

Appendix C. XSD and XML documents

The full XSD model derived from the UML model in Figure 17 is shown below.

```
<?xml version = "1.0" encoding = "utf-8"?>
<xsd:schema elementFormDefault = "qualified" xmlns:xs =
"http://www.w3.org/2001/XMLSchema">
    <xsd:element name = "jaguarlandrover"  type = "Company" />
    <xsd:complexType name = "Company">
        <xsd:sequence>
            <xsd:element name = "vehicle" type = "Vehicle" minOccurs =
"1" maxOccurs = "unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name = "Vehicle">
        <xs:sequence>
            <xs:element name = "colour" type = "xsd:string" minOccurs
= "1" maxOccurs = "1"/>
            <xs:element name = "transmissionType" type = "xsd:string"
minOccurs = "1" maxOccurs = "1"/>
            <xs:element name = "body" type = "Body" minOccurs = "1"
maxOccurs = "2"/>
            <xs:element name = "engine" type = "Engine" minOccurs =
"1" maxOccurs = "unbounded"/>
        </xs:sequence>
        <xs:attribute name = "vehicleId" type = "xsd:string"/>
    </xsd:complexType>

    <xsd:complexType name = "Body">
        <xsd:sequence>
            <xsd:element name = "numberOfDoors" type = "xsd:integer"
minOccurs = "1" maxOccurs = "unbounded"/>
        </xsd:sequence>
        <xs:attribute name = "bodyType" type = "xsd:string"/>
    </xsd:complexType>

    <xsd:complexType name = "Engine">
        <xsd:sequence>
            <xsd:element name = "fuelType" type = "xsd:string"
minOccurs = "1" maxOccurs = "1"/>
        </xsd:sequence>
        <xs:attribute name = "engineSize" type = "xsd:integer"/>
    </xsd:complexType>

    <xsd:complexType name = "vehicleType">
        <xsd:complexContent>
            <xsd:extension base = "Vehicle">
                <xsd:sequence>
                    <xsd:element name = "brandType" type = "DriveType"
minOccurs = "1" maxOccurs = "1"/>
                    <xsd:element name = "wheelDrive" type =
"xsd:integer" minOccurs = "1" maxOccurs = "1"/>
                    <xsd:element name = "driveType" type = "DriveType"
minOccurs = "1" maxOccurs = "1"/>
                </xsd:sequence>
```

```
                <xs:attribute name = "vehicleBrand" type =
"xsd:string"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name = "DriveType">
        <xsd:complexContent>
            <xsd:extension base = "VehicleType">
                <xsd:sequence>
                    <xsd:element name = "driveType" type =
"xsd:string" minOccurs = "1" maxOccurs = "1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

The full XML document derived from the UML and XSD is shown below.

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<jaguarlandrover
    xmlns = "http://dvi.org/DVI"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance">
    <vehicle vehicleId = "ER234678">
        <colour>White</colour>
        <transmissionType>Auto</transmissionType>
        <body bodyType = "Aluminium">
            <numberOfDoors>5</numberOfDoors>
        </body>
        <engine engineSize = "3">
            <fuelType>Diesel</fuelType>
        </engine>
    </vehicle>
     <vehicle vehicleId = "XJ234678" vehicleBrand= "Jaguar" xsi:type =
"VehicleType">
        <colour>Red</colour>
        <transmissionType>Manual</transmissionType>
        <body bodyType = "Aluminium">
            <numberOfDoors>5</numberOfDoors>
        </body>
         <engine engineSize = "3">
            <fuelType>Diesel</fuelType>
        </engine>
        <brandType>XJ</brandType>
        <wheelDrive>2</wheelDrive>
        <driveType>RightHandDrive</driveType>
    </vehicle>
</jaguarlandrover>
```

Appendix D. Attributes used for the development of SDVM

Table 28 shows the list of attributes used for the design of the new Standardised Design Verification Method (SDVM).

Test Specification Attributes	
Vehicle network I/O interfaces	• CAN • LIN • MOST • Ethernet • Other (this may include, USB, hands free phone connectivity, iPod, voice, Bluetooth, DVD, dual view touch screen, AM/FM, 3G, 4G)
Vehicle hardware I/O interfaces	• Digital I/O such as switch inputs and outputs (low/high side driver) • Analogue I/O such as resistive voltage inputs and voltage control outputs • Pulse Width Modulation (PWM), a common type of digital I/O often used in the automotive industry. A typical application that uses PWM is window control
ECU specific monitoring parameters	• Read of Electrically Erasable Programmable Read-Only Memory (EEPROM). A typical application that uses EEPROM read is vehicle seat position control • Read of calibration parameters such as vehicle engine maps • Other various ECU software parameters • Read and write of car configuration. It is used to configure a vehicle for different markets (i.e. left or right hand drive) • Vehicle diagnostics. Mainly used to put ECU into programing mode or read fault codes after a cycle of tests
Type of validation	• ON/OFF. An ON/OFF input signal that causes the output to change state. This is a common attribute for body, comfort and climate systems • Function-based. The input signal is a function such as ramp up and rump down. This is a common attribute for chassis and powertrain systems • IF/ELSE, the input or output signal is subject to an IF and ELSE condition. This is a common attribute for infotainment systems • FOR LOOP the input or output signal is subject to a FOR LOOP condition. This is a common attribute for infotainment systems • Truth table. The input and output signals are subject to a truth table. This is a common attribute for infotainment systems • Lookup table. The input and output signals are subject to a lookup table. This is a common attribute for transmission systems
Control logic of the embedded software under test	• State control logic. This is a common attribute for body, comfort and climate systems • Continuous control logic. This is a common attribute for chassis and powertrain systems • Event driven control logic. This is a common attribute for infotainment and HMI systems
Test target or test platform	• Offline PC-based • Real-time
Test method	• Data are confidential and therefore are only available in Submission 4

Table 28. List of attributes used for the design of SDVM.

Appendix E. Supported conditions and branches within the SDVM

Table 29 provides full description of the supported conditions and branches within the SDVM.

Condition/ Branch	SDVM Template Semi-formal Notation - Example	Description
For Loop	LOOP(5|)	Loops five times the test case
While Loop	LOOP(PowerModeSignal>0)	Loops until the condition is satisfied
If Else	CHECKIF(PowerModeSignal==0)	Proceed If condition is satisfied
Goto	GOTO(LOC_TEST_ID_1)	Go to test case with ID 1

Table 29. Description of condition and branches.

An example of a condition/branch is given as follows:

Test case condition using informal specification:

Check if vehicle signal called *"PowerMode_MS"* is greater than 6 (6 = Ignition). If this condition is true then move to test case with ID 4.

Test case condition in condition/braches section using semi-formal specification:

CHECKIF(PowerMode_MS>6) {GOTO(TEST_CASE_ID_4)}

Appendix F. FURPS+ model

Table 30 shows the definition of FURPS+ requirements. Functionality is the only type of functional requirement. Functionality is referring to what the customer wants and as a result the main features of the product. In this research Functionality refers to a set of requirements to deliver the main features of PITS. The remaining types of requirements in the FURPS+ model are of non-functional type.

FURPS	**Meaning**	**Type**
Functionality	Main product features.	Functional
Usability	User interface, accessibility, aesthetics and consistency.	Non-functional
Reliability	Availability, accuracy and recovery.	
Performance	Throughput, response time, recovery time and start-up time.	
Supportability	Testability, adaptability, maintainability, compatibility, configurability, installability and scalability.	
+	**Meaning**	
Design requirements	Design constraints.	
Implementation requirements	Coding or construction constraints.	
Interface requirements	Interaction with an external item.	
Physical requirements	Physical constraint imposed on the hardware.	

Table 30. FURPS+ requirements classification.

Appendix G. Summary of evaluation results

G.1. Satisfaction of MBPE process requirements

Number	Requirement	Satisfied	How
1	Standardised	Y	Defined in Engineering Standard A
2	Tool independent	Y	All levels are tool independent as defined in Engineering Standard A
3	Link and integration to JLR PCS	Y	All levels have link to JLR PCS as defined in Engineering Standard A
4	Support of offline (PC) model-based automated testing	Y	Supported by levels 0, 1 and 2 in Engineering Standard A
5	Support of real-time model-based automated testing	Y	Supported by levels 2, 3, 4 and 5 in Engineering Standard A, Design Rule A and Design Rule B
6	Support of functional/non-functional and vehicle diagnostics testing	Y	Supported by all levels as defined in Engineering Standard A
7	Design verification methods abstraction to support different vehicle levels	Y	Supported by levels 1, 3, 4 and 5 in Engineering Standard A
8	Support of test data exchange	Y	As defined in the Design Verification Interface (DVI)
9	Support of product complexity	Y	As defined in the MBPE deployment strategy (Submission 2)
10	Support of agile and concurrent engineering	Y	All levels are iterative as defined in Engineering Standard A
11	Re-use of executable functional models via model-based design	Y	Supported by levels 0 and 1 in Engineering Standard A
12	Support of quality, safety and software related industry standards	Y	Support of ISO 14229, ISO 26262, MCDC, MISRA, CMMI and SPICE.
13	Automated generation and re-use of test scripts via common design verification interface	Y	As defined in the Design Verification Interface (DVI) and Standardised Design Verification Method (SDVM)
14	Software verification and validation prior to supplier release	Y	Supported by levels 0, 1 and 2 in Engineering Standard A
15	Support vehicle level model integration prior to production software and hardware	Y	Supported by levels 0, 1 and 2 in Engineering Standard A
16	Support customer and system requirements validation	Y	Supported by levels 0 and 1 in Engineering Standard A
17	Traceability between different levels of abstraction and models	Y	Defined in Engineering Standard B
18	Support of OEM/Supplier information management	Y	Defined in Engineering Standard A and JLR software Statement of Work (SoW)
19	Integrated deployment strategy for robust introduction within a large organisation	Y	As defined in the MBPE deployment strategy (Submission 2)

Table 31. Summary of MBPE requirements satisfaction.

G.2. Capability of MBPE process

Table 32 summarises the MBPE process Levels failure modes and/or software defects detection capability. Detection capability is rated as L (Low, no detection or <10% opportunity to detect), M (Medium, <60% opportunity to detect) and H (High, <100% opportunity to detect). Detection capability across all levels is incremental and hence each Level requires the execution of the previous level.

Areas for Potential Failure Modes and/or Software Defects	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Customer Requirements	H	M	M	M	H	H
System Requirements	H	H	M	M	M	H
Software Specification	L	H	H	H	H	H
Arithmetic	L	L	M	H	H	H
Programming	L	L	H	H	H	H
Compiler	L	L	M	H	H	H
Scheduler	L	L	M	H	H	H
System Integration	M	L	M	M	H	H
Software I/O Compatibility	L	H	H	H	H	H
Hardware I/O Compatibility	L	L	M	H	H	H
Diagnostic Services (Part I)	L	L	L	H	H	H
Diagnostics Part II (Including DTCs)	L	L	L	M	M	H
Distributed Functionality	M	L	M	M	H	H
Immunity to Low Voltage	L	L	L	H	H	H
Energy and Load Management	M	L	L	M	M	H
Network Management	M	L	M	M	M	H
Network Robustness	L	L	M	M	M	H
Car Configuration	M	M	M	H	H	H
Memory / EEPROM	L	L	L	H	H	H
Manufacturing Validation	M	M	L	H	H	H
Software Download Functionality	L	L	L	M	M	H
Gateway Functionality	M	M	M	M	H	H
Start Up & Shut Down Vehicle Functionality	M	L	M	L	M	H
Closed Loop Energy Flow	M	L	M	L	M	H

Table 32. Failure modes and/or software defects detection capability (L = Low, M = Medium, H = High) across all MBPE levels.

G.3. Satisfaction of DVI requirements and comparison with other standards

Table 33 summarises the DVI requirements satisfaction.

Number	Requirement	Satisfied	How
1	Support of different signal profiles	Y	Proven through PITS application and SDVM
2	Control flow (Logical and conditional)	Y	Proven through PITS application and SDVM
3	Support functional- based type testing	Y	Proven through PITS application and SDVM
4	Support offline testing (MIL/SIL-based)	Y	Proven through Locking and Wiper control use cases
5	Support real-time testing (RCP/HIL-based)	Y	Proven through Locking and Wiper control use cases
6	Support in-vehicle networks diagnostics testing	Y	Proven through PITS application and SDVM
7	Support all MBPE abstraction levels	Y	Proven through SDVM requirements satisfaction
8	Link to standardised DVM	Y	Proven through PITS application and SDVM
9	Support auto-generation of test cases	Y	Proven through Locking and Wiper control use cases
10	Plugin for MATLAB interface – PC-based testing	Y	Proven through Locking and Wiper control use cases
11	Plugin for Python (dSPACE) interface – Real-time based testing	Y	Proven through Locking and Wiper control use cases
12	Proven with real automotive applications	Y	Proven through Locking and Wiper control use cases

Table 33. Summary of DVI requirements satisfaction.

Table 34 show a comparison between the DVI and other automotive related standards.

Number	Requirement	OTX	ODX	FMI	ATX	DVI
1	Support of different signal profiles	N	N	N	Y	Y
2	Control flow (Logical and conditional)	Y	N	N	N	Y
3	Support functional- based type testing	Y	N	P	Y	Y
4	Support offline testing (MIL/SIL-based)	N	N	Y	Y	Y
5	Support real-time testing (RCP/HIL-based)	Y	N	P	Y	Y
6	Support in-vehicle networks diagnostics testing	N	N	N	N	Y
7	Support all MBPE abstraction levels	N	N	P	P	Y
8	Link to standardised DVM	N	N	N	P	Y
9	Support auto-generation of test cases	N	P	N	P	Y
10	Plugin for MATLAB interface – PC-based testing	N	N	N	N	Y
11	Plugin for Python (dSPACE) interface – Real-time based testing	N	N	N	N	Y
12	Proven with real automotive applications	Y	Y	P	N	Y
Number	**Additional Requirement Driven by Other Standards**	**OTX**	**ODX**	**FMI**	**ATX**	**DVI**
13	Support ECU diagnostics testing	Y	Y	N	N	N
14	Simulation models integration / Co-simulation	N	N	Y	N	N
15	International standard	Y	Y	N	N	N

Table 34. DVI comparison against other standards.

Key:

Y = Supported

N = Not Supported

P = Partially Supported

G.4. Vehicle systems/features written in the proposed SDVM

Table 35 shows the list of key vehicle systems/features written in the proposed SVDM. More details are given in Submission 6.

Vehicle System/Feature	MBPE Process	Implemented	Reference
Interior lights	Level 4	Y	Appendix A in Submission 6
Transmission system	Level 3	Y	Appendix A in Submission 6
Comfort system	Level 3	Y	Appendix A in Submission 6
Climate control	Level 3	Y	Appendix A in Submission 6
Hybrid control	Level 1/2	Y	Appendix A in Submission 6
Electronic park brake control	Level 4	Y	Appendix A in Submission 6
Human machine interface	Level 3/4	Y	Appendix A in Submission 6
Infotainment	Level 4/5	Y	Appendix A in Submission 6
Engine management system	Level 3/4	Y	Appendix A in Submission 6
Power supply	Level 5	Y	Appendix A in Submission 6
Locking system	Level 5	Y	Appendix A in Submission 6
Wipers system	Level 0/1	Y	Appendix A in Submission 6

Table 35. Vehicle systems/features written in the SDVM.

Key:

Y - Fully implemented

P - Partially implemented

N - Not implemented

G.5. A sample of PITS requirements satisfaction

Table 36 shows a sample of PITS stakeholder, functional and non-functional requirements satisfaction. The full list can be found in Submission 6.

Number	Requirement	Type	How
STK_SRS_1	The system shall provide an interface for the creation and automated execution of test cases suitable for all Model Based Product Engineering (MBPE) levels (Level 1 to Level 5).	Stakeholder	Implemented through SDVM.
STK_SRS_2	The system shall automatically generate platform independent test scripts directly from the SDVM.	Stakeholder	Implemented in the DVI XML.
STK_SRS_3	The system shall support a generic DVI interface suitable for test exchange and integration with different test automation environments (support for offline and real-time test targets).	Stakeholder	Implemented through MATLAB (offline) and dSPACE (real-time) plugins.
FR_SRS_01	The PIT system shall generate a generic XML representation of the DVM test data as an output.	Functional	Implemented in the SDVM and DVI.xml.
FR_SRS_02	The DVI generic XML data shall be the intermediate representation of the test data.	Functional	Implemented in the DVI.xml.
FR_SRS_03	The DVI generic XML test data shall be the input for the auto-generated platform dependent test script.	Functional	Implemented in the DVI.xml.
NFR_SRS_01	The PIT system shall generate an XML representation of the SDVM test data as an output.	Non-functional	Implemented in the DVI.xml.
NFR_SRS_02	The DVI XML test data shall be the intermediate representation of the test data.	Non-functional	Implemented in the DVI.xml.
NFR_SRS_03	The generic DVI XML shall be developed to support the Engineering Standard A.	Non-functional	Implemented in the SDVM and DVI.xml.

Table 36. Sample of PITS requirements satisfaction.

G.6. Implementation of signal profiles defined in the SDVM

Table 37 shows an offline and real-time implementation of signal profiles defined in the SDVM. Two signal profiles were not implemented in real-time due to dSPACE constraints.

Signal Name in the SDVM	Signal Profile Description	Offline Implementation (MATLAB)	Real-time Implementation (dSPACE)
Sig_RAMP	Ramp.	Y	Y
Sig_CHIRP_SIG	Chirp.	Y	Y
Sig_TIME_BASED_PULSE	Time based pulse.	Y	Y
Sig_STEP	Step.	Y	Y
Sig_CONSTANT	Constant value.	Y	Y
Sig_UNIFORM_RAND_NUM	Uniform random number.	Y	Y
Sig_SIN	Sinusoidal.	Y	Y
Sig_SINGLE_PULSE	Single pulse.	Y	Y
Sig_REPEATED_SEQ	Repeated sequence.	Y	N (dSPACE real-time target does not provide all parameters)
Sig_SAMPLE_BASED_PULSE	Sample based pulse.	Y	Y
Sig_SIGNAL_GEN	Signal generator.	Y	Y
Sig_SIGNAL_BUILDER	Signal builder.	Y	N (dSPACE real-time target does not provide all parameters)

Table 37. Signal profiles defined in the SDVM and implementation in MATLAB and dSPACE through PITS.

Key:

Y - Fully implemented

P - Partially implemented

N - Not implemented

G.7. Vehicle remote key locking control system SIMULINK model

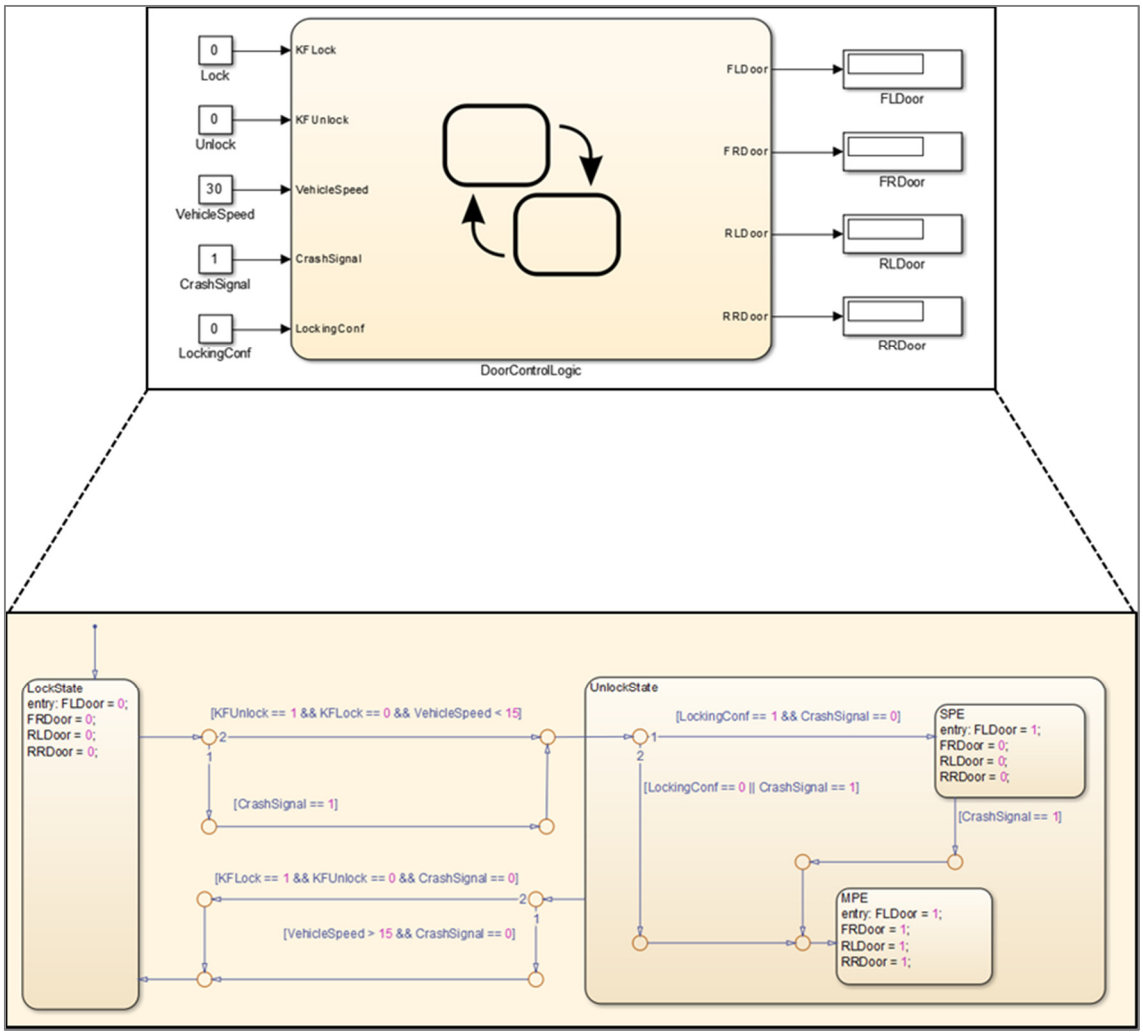


Figure 44. Typical vehicle remote key locking control system SIMULINK model.

G.8. Vehicle remote key locking system test definition in SDVM

Test ID	Purpose of the Test	Test Sequence	I/O	I/O Description	Condition /Branches	Input Signal			Output Signal		
						Name	Type	Value	Name	Type	Value
LOC_TEST_ID_1	The purpose of the test is to test vehicle doors locking functionality. Step1: Multipoint entry configuration (all doors must unlock at a lock command request). Step2: Lock the vehicle via the key fob lock button. Check all four doors are locked.	Step 1	Input(s)	Set Configuration to Mutipoint Entry		LockingConf	CONSTANT	0			
		Step 2		Reset Unlock		Unlock	CONSTANT	0			
		Step 3		Set Lock		Lock	SINGLE_PULSE	(1,2,0)			
		Step 4		Vehicle Speed set to 0 mph		VehicleSpeed	CONSTANT	0			
		Step 5		Reset Crash Signal to 0		CrashSignal	CONSTANT	0			
		Step 6	Output(s)	Check Front Left Door is Locked					FLDoor	CONSTANT	0
		Step 7		Check Front Right Door is Locked					FRDoor	CONSTANT	0
		Step 8		Check Rear Left Door is Locked					RLDoor	CONSTANT	0
		Step 9		Check Rear Right Door is Locked					RRDoor	CONSTANT	0
LOC_TEST_ID_2	The purpose of the test is to test vehicle doors unlocking functionality. Step1: Multipoint entry configuration (all doors must unlock at a lock command request). Step2: Unlock the vehicle via the key fob unlock button. Check all four doors are unlocked.	Step 1	Input(s)	Set Configuration to Mutipoint Entry		LockingConf	CONSTANT	0			
		Step 2		Reset Unlock		Unlock	SINGLE_PULSE	(1,2,1)			
		Step 3		Set Lock		Lock	CONSTANT	0			
		Step 4		Vehicle Speed set to 0 mph		VehicleSpeed	CONSTANT	0			
		Step 5		Reset Crash Signal to 0		CrashSignal	CONSTANT	0			
		Step 6	Output(s)	Check Front Left Door is Unlocked					FLDoor	STEP	(1,0,1)
		Step 7		Check Front Right Door is Unlocked					FRDoor	STEP	(1,0,1)
		Step 8		Check Rear Left Door is Unlocked					RLDoor	STEP	(1,0,1)
		Step 9		Check Rear Right Door is Unlocked					RRDoor	STEP	(1,0,1)
LOC_TEST_ID_3	The purpose of the test is to test vehicle doors locking functionality. Step1: Multipoint entry configuration (all doors must unlock at a lock command request). Step2: Lock the vehicle via the key fob lock button. Check all four doors are locked.	Step 1	Input(s)	Set Configuration to Mutipoint Entry		LockingConf	CONSTANT	0			

Figure 45. Sample of vehicle remote key locking system test definition in SDVM.

G.9. Results from offline automated testing

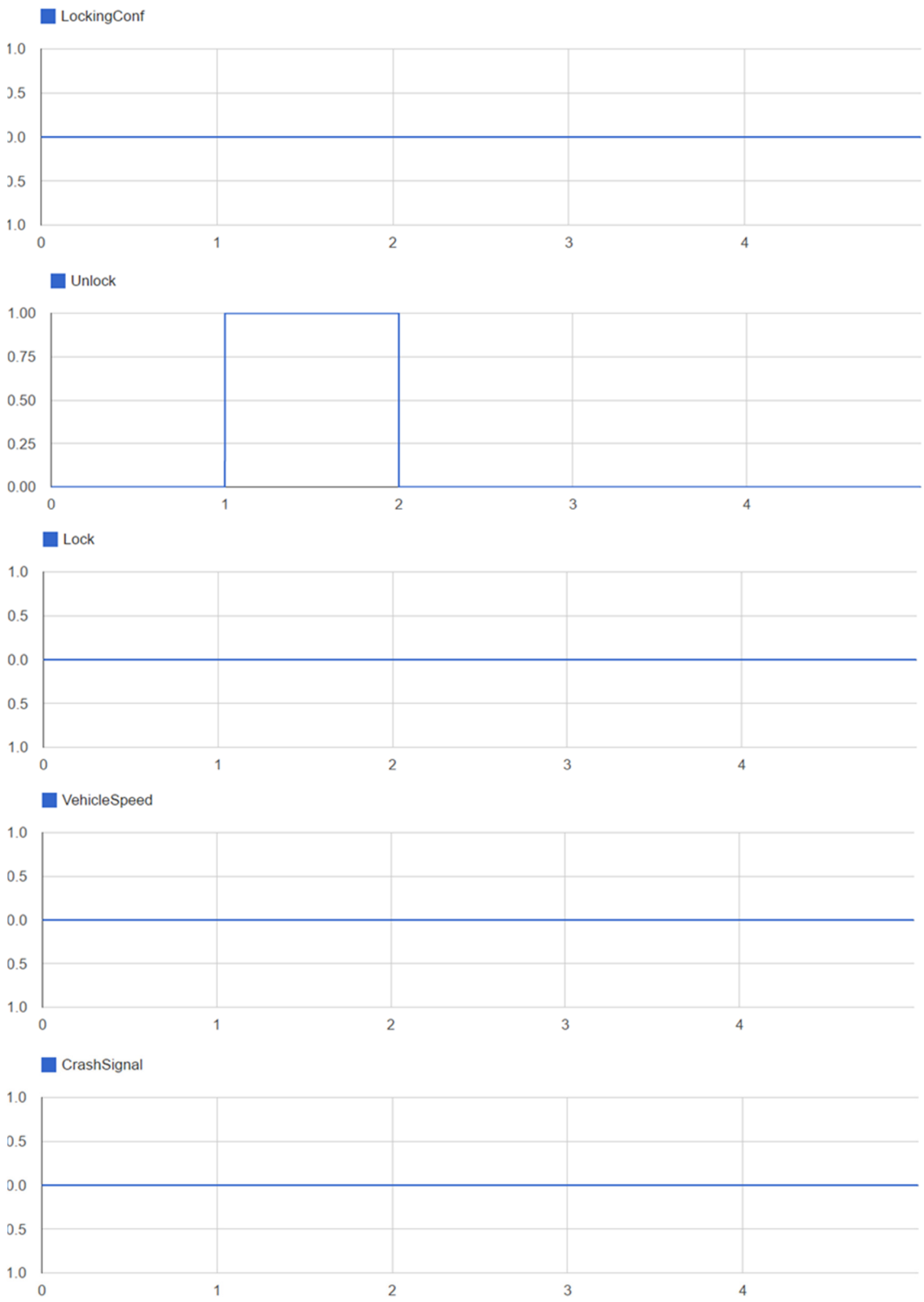


Figure 46. Input signals for vehicle remote key locking – LOC_TEST_ID_2.

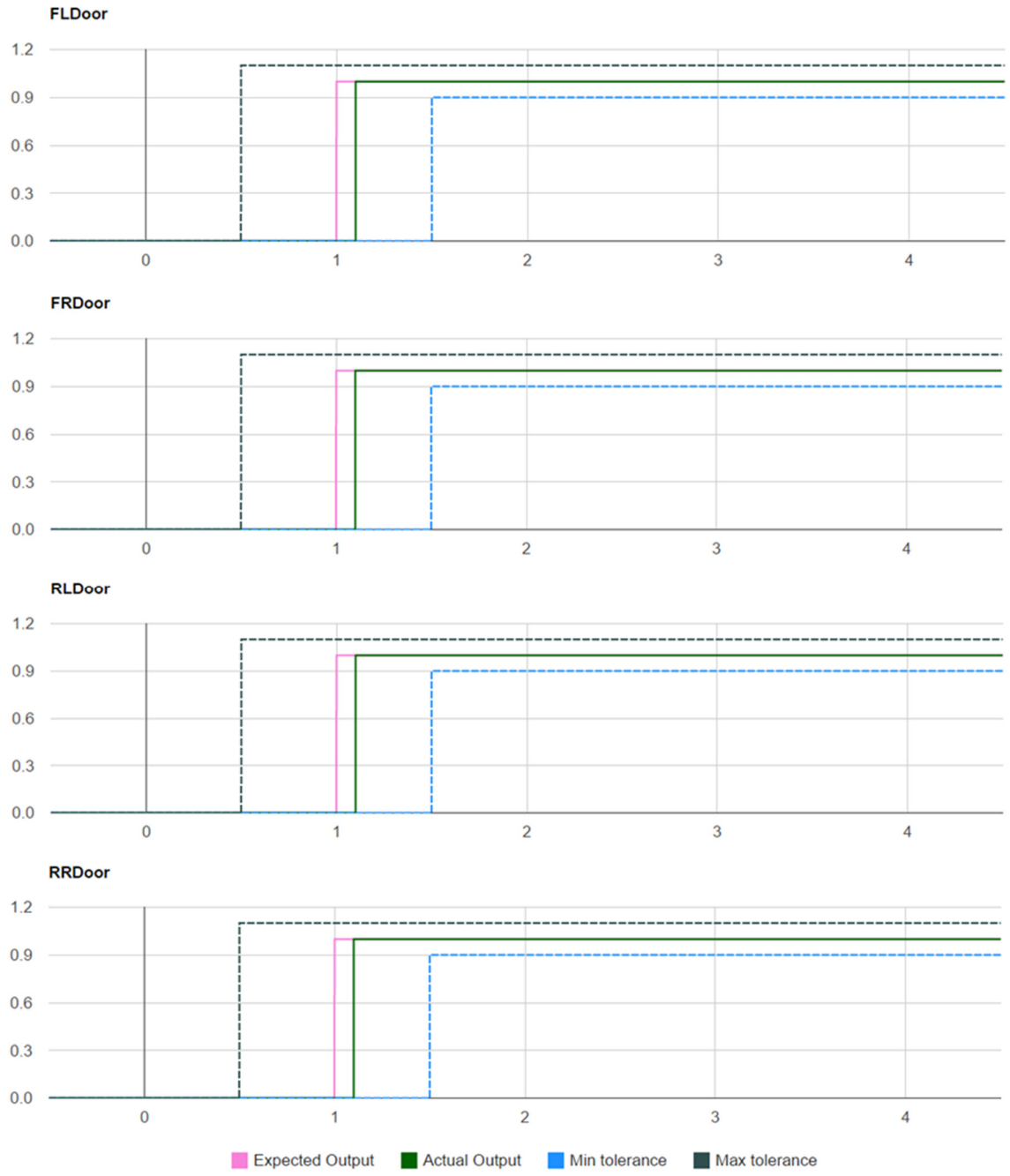


Figure 47. Actual and expected signals for vehicle remote key locking – LOC_TEST_ID_2.

G.10. Results from real-time automated testing

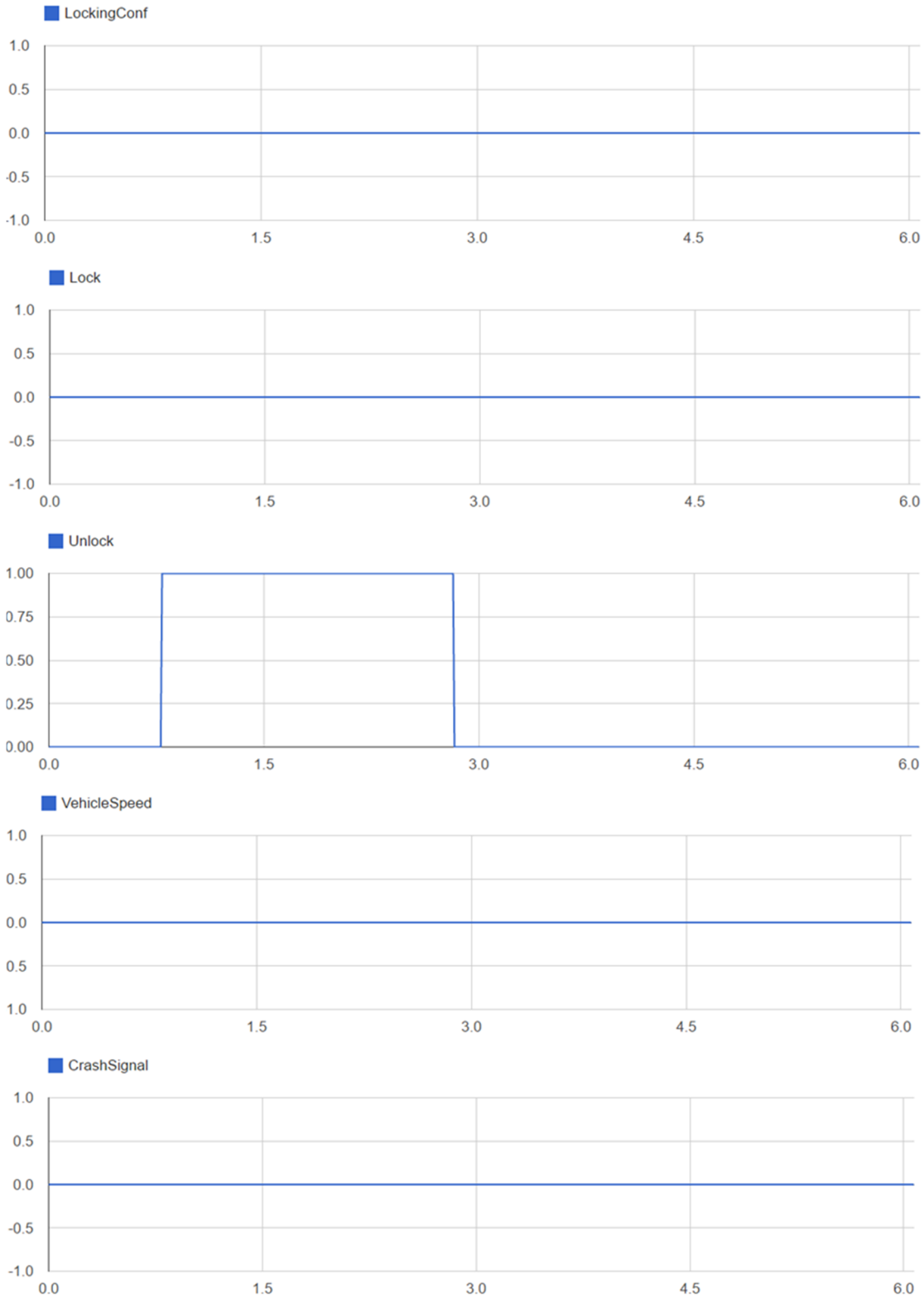


Figure 48. Real-time input signals for vehicle remote key locking system – LOC_TEST_ID_2.

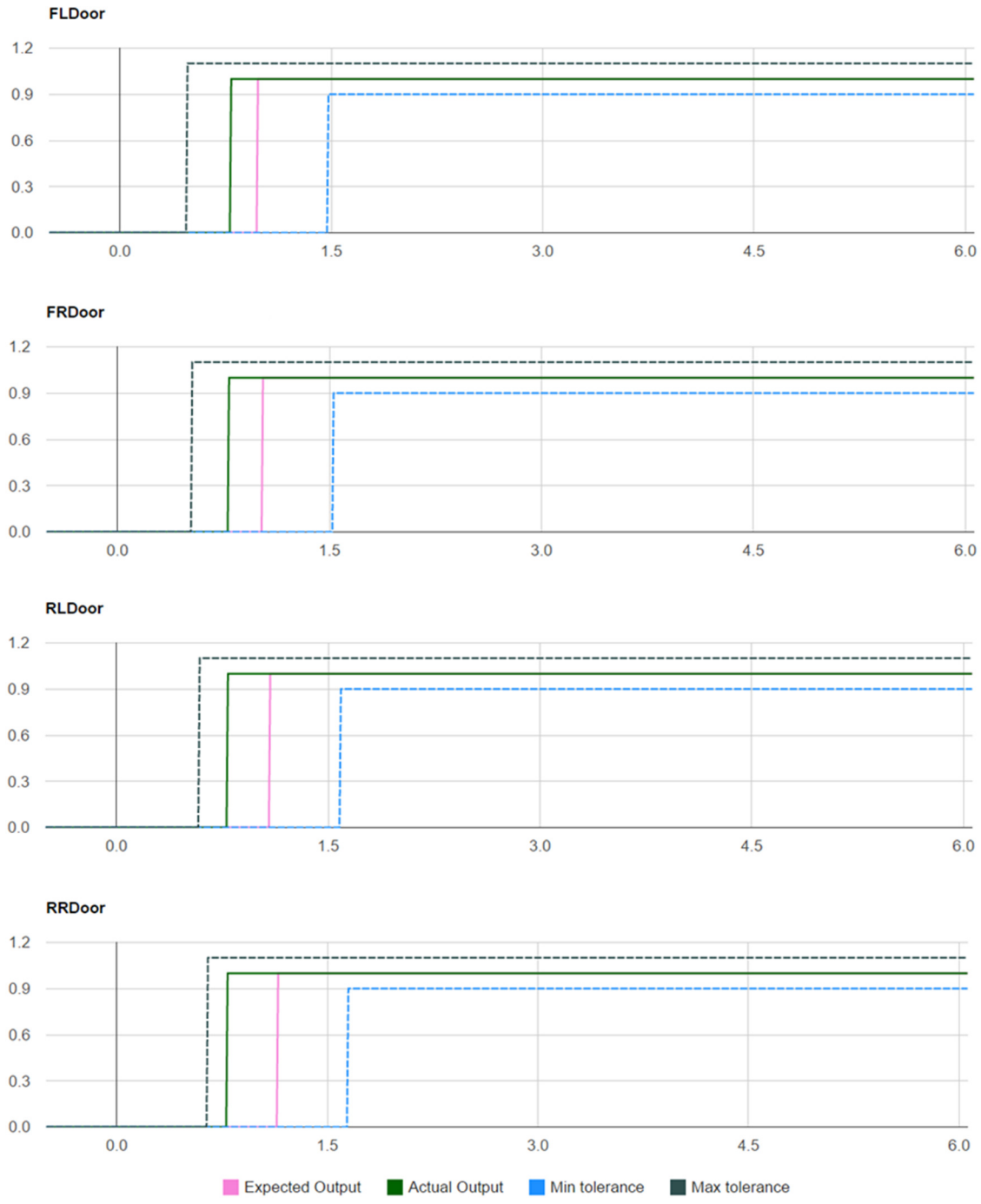


Figure 49. Real-time actual and expected signals for vehicle remote key locking system – LOC_TEST_ID_2.